

# **Introduction to Geoprocessing Scripts Using Python®**

Student Edition

Copyright © 2012 Esri  
All rights reserved.

Course version 5.1. Version release date June 2012.

Printed in the United States of America.

The information contained in this document is the exclusive property of Esri. This work is protected under United States copyright law and other international copyright treaties and conventions. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or by any information storage or retrieval system, except as expressly permitted in writing by Esri. All requests should be sent to Attention: Contracts and Legal Services Manager, Esri, 380 New York Street, Redlands, CA 92373-8100 USA.

**EXPORT NOTICE:** Use of these Materials is subject to U.S. export control laws and regulations including the U.S. Department of Commerce Export Administration Regulations (EAR). Diversion of these Materials contrary to U.S. law is prohibited.

The information contained in this document is subject to change without notice.

### US Government Restricted/Limited Rights

Any software, documentation, and/or data delivered hereunder is subject to the terms of the License Agreement. The commercial license rights in the License Agreement strictly govern Licensee's use, reproduction, or disclosure of the software, data, and documentation. In no event shall the US Government acquire greater than RESTRICTED/LIMITED RIGHTS. At a minimum, use, duplication, or disclosure by the US Government is subject to restrictions as set forth in FAR §52.227-14 Alternates I, II, and III (DEC 2007); FAR §52.227-19(b) (DEC 2007) and/or FAR §12.211/12.212 (Commercial Technical Data/Computer Software); and DFARS §252.227-7015 (DEC 2011) (Technical Data - Commercial Items) and/or DFARS §227.7202 (Commercial Computer Software and Commercial Computer Software Documentation), as applicable. Contractor/Manufacturer is Esri, 380 New York Street, Redlands, CA 92373-8100, USA.

@esri.com, 3D Analyst, ACORN, Address Coder, ADF, AML, ArcAtlas, ArcCAD, ArcCatalog, ArcCOGO, ArcData, ArcDoc, ArcEdit, ArcEditor, ArcEurope, ArcExplorer, ArcExpress, ArcGIS, ArcGlobe, ArcGrid, ArcIMS, ARC/INFO, ArcInfo, ArcInfo Librarian, ArcLessons, ArcLocation, ArcLogistics, ArcMap, ArcNetwork, *ArcNews*, ArcObjects, ArcOpen, ArcPad, ArcPlot, ArcPress, ArcPy, ArcReader, ArcScan, ArcScene, ArcSchool, ArcScripts, ArcSDE, ArcSdl, ArcSketch, ArcStorm, ArcSurvey, ArcTIN, ArcToolbox, ArcTools, ArcUSA, *ArcUser*, ArcView, ArcVoyager, *ArcWatch*, ArcWeb, ArcWorld, ArcXML, Atlas GIS, AtlasWare, Avenue, BAO, Business Analyst, Business Analyst Online, BusinessMAP, CityEngine, CommunityInfo, Database Integrator, DBI Kit, EDN, Esri, Esri—Team GIS, Esri—*The GIS Company*, Esri—The GIS People, Esri—The GIS Software Leader, FormEdit, GeoCollector, Geographic Design System, Geography Matters, Geography Network, GIS by Esri, GIS Day, GIS for Everyone, GISData Server, JTX, MapIt, Maplex, MapObjects, MapStudio, ModelBuilder, MOLE, MPS—Atlas, PLTS, Rent-a-Tech, SDE, SML, Sourcebook·America, SpatialLABS, Spatial Database Engine, StreetMap, Tapestry, the ARC/INFO logo, the ArcGIS logo, the ArcGIS Explorer logo, the ArcPad logo, the Esri globe logo, the Esri Press logo, the GIS Day logo, the MapIt logo, The Geographic Advantage, The Geographic Approach, The World's Leading Desktop GIS, *Water Writes*, www.arcgis.com, www.esri.com, www.geographynetwork.com, www.gis.com, www.gisday.com, and Your Personal Geographic Information System are trademarks, service marks, or registered marks in the United States, the European Community, or certain other jurisdictions. CityEngine is a registered trademark of Procedural AG and is distributed under license by Esri.

Other companies and products or services mentioned herein may be trademarks, service marks or registered marks of their respective mark owners.

## Course introduction

Introduction .....	i
Course goals .....	i
Additional resources .....	i
Installing the course data .....	ii

## 1 Running scripts in Python

Lesson introduction .....	1-1
Integrated Development Environment (IDE) .....	1-2
Running scripts in Python window .....	1-3
Exercise 1A: Use the PythonWin IDE (Instructor-led) .....	1-5
Open and configure PythonWin .....	1-6
Run the script .....	1-6
The ArcPy site package .....	1-8
ArcPy functions and classes .....	1-9
The ArcPy modules .....	1-11
Choosing a scripting environment .....	1-13
Tips and best practices .....	1-14
Exercise 1B: Run scripts in Python .....	1-15
Buffer schools in Python window .....	1-16
Update script in PythonWin .....	1-18
Verify results in ArcMap .....	1-20
Lesson review .....	1-21

## 2 Describing data

Lesson introduction .....	2-1
The Describe function .....	2-2
Generic Describe object properties .....	2-3
Feature class Describe properties .....	2-4
Raster Describe properties .....	2-7
Describing a feature class and raster .....	2-9
Fill-in-the-blank .....	2-10
Exercise 2: Work with the Describe object .....	2-13
Describe a feature class and a geodatabase .....	2-14
Describe a list of feature classes .....	2-15
Clip raster datasets with Describe object properties .....	2-17
Challenge: Describe dataset and coordinate system properties .....	2-21
Lesson review .....	2-22

## 3 Automating scripts with lists

Lesson introduction .....	3-1
The ArcPy List functions .....	3-2
Explore the ArcPy List functions .....	3-5

Working with List functions .....	3-6
List data .....	3-7
Exercise 3: Automate scripts with the ArcPy List functions .....	3-9
List field properties .....	3-11
Buffer feature classes .....	3-13
Verify script results .....	3-15
Lesson review .....	3-16

## 4 Working with Selections

Lesson introduction .....	4-1
Selection tools in ArcMap .....	4-2
Terms commonly used when working with selections .....	4-3
Tools that accept Feature Layers .....	4-4
Working with a selection .....	4-6
The MakeFeatureLayer tool .....	4-7
The FieldInfo object .....	4-9
The AddFieldDelimiters function .....	4-11
Determining a workflow .....	4-12
Creating feature layer and get feature count .....	4-13
Exercise 4: Work with Feature Layers and Selections .....	4-15
Create new script .....	4-16
Create Feature Layers .....	4-17
Perform Spatial Selection .....	4-18
Create Feature Class from selection .....	4-19
Lesson review .....	4-20

## 5 Working with Cursors

Lesson introduction .....	5-1
The arcpy.da cursors .....	5-2
Using the SearchCursor .....	5-4
Using the UpdateCursor .....	5-7
Using the InsertCursor .....	5-9
Tips and best practices for arcpy.da Cursors .....	5-11
Exercise 5: Work with cursors .....	5-13
Research the da Cursors .....	5-14
Work with the da.SearchCursor .....	5-14
Work with the da.UpdateCursor .....	5-16
Work with the da.InsertCursor .....	5-17
Lesson review .....	5-20

## 6 Working with Geometry objects

Lesson introduction .....	6-1
Creating geometry objects .....	6-2
Creating Point objects .....	6-3

Creating Polyline geometry objects .....	6-7
Creating Polygon geometry objects .....	6-9
The geometry object.....	6-11
Constructing multipart geometry .....	6-13
Constructing multipart polygons.....	6-14
Creating and updating feature geometry .....	6-15
Using geometry object with geoprocessing tool .....	6-16
Accessing geometry objects .....	6-17
Exercise 6: Work with geometry objects and cursors.....	6-19
Create geometry objects .....	6-20
Access Shape geometry .....	6-22
Update existing features .....	6-23
Create new features .....	6-25
(Optional) Use geometry object with geoprocessing tool .....	6-27
Lesson review .....	6-29

## 7 Sharing scripts

Lesson introduction .....	7-1
Terms commonly used when sharing scripts .....	7-2
Scripting advantages in ArcMap .....	7-3
Making scripts dynamic.....	7-4
Creating and sharing a script tool .....	7-5
Advantages of attaching a script to a custom tool.....	7-6
Using the Add Script wizard .....	7-7
Running a script.....	7-10
Sharing a script.....	7-12
Plan out a project .....	7-16
Exercise 7: Share scripts through geoprocessing packages.....	7-21
Add pseudocode to the script.....	7-22
Write script code.....	7-22
Attach script to custom tool.....	7-24
Run script in ArcMap.....	7-25
Share results as a geoprocessing package .....	7-26
Verify the geoprocessing package.....	7-27
Lesson review .....	7-29

## 8 Debugging scripts and handling runtime errors

Lesson introduction .....	8-1
Debugging workflow.....	8-2
Debugging scripts in PythonWin.....	8-3
Handling script exceptions.....	8-5
Working with exceptions.....	8-6
Using Exception as e .....	8-6
Using arcpy.ExecuteError .....	8-7

Using the traceback module .....	8-8
Getting geoprocessing messages .....	8-9
Tips and best practices.....	8-11
Exercise 8: Debug scripts and handle exceptions.....	8-13
Debug the script .....	8-14
Incorporate a try-except block.....	8-14
Incorporate an Exception as e statement .....	8-16
Incorporate an arcpy.ExecuteError exception class .....	8-18
Use the traceback module .....	8-19
Challenge: Challenge .....	8-22
Lesson review .....	8-23

## 9 Automating map production

Lesson introduction .....	9-1
The arcpy.mapping module .....	9-2
Terms commonly used when working with the arcpy.mapping module .....	9-3
Using the arcpy.mapping module to modify map document contents .....	9-4
Referencing the map document.....	9-5
Listing map document contents.....	9-7
Managing layers .....	9-8
Managing layout elements .....	9-10
Publishing maps .....	9-11
Exporting maps and reports.....	9-13
Exercise 9: Work with map document contents .....	9-15
Access map document in ArcMap .....	9-16
Access map document in script .....	9-16
Update layer symbology .....	9-17
Update layout text elements.....	9-19
Verify changes in ArcMap .....	9-19
Challenge: Automate script for multiple mxds .....	9-21
Lesson review .....	9-22

## Appendixes

Appendix A: Esri data license agreement.....	A-1
Appendix B: Answers to lesson review questions	
Lesson 1: Running scripts in Python.....	B-1
Lesson 2: Describing data.....	B-2
Lesson 3: Automating scripts with lists .....	B-3
Lesson 4: Working with Selections.....	B-4
Lesson 5: Working with Cursors.....	B-5
Lesson 6: Working with Geometry objects .....	B-6
Lesson 7: Sharing scripts.....	B-7
Lesson 8: Debugging scripts and handling runtime errors.....	B-8
Lesson 9: Automating map production .....	B-9

# Introduction

This course uses Python's language syntax and modules to access and automate geoprocessing workflows through Python scripts. These scripts can be run inside of the Python Integrated Development Environment, within the Python window in ArcGIS Desktop, as custom tools within ArcToolbox, or shared as geoprocessing packages.

This course covers the ArcPy site package, which integrates Python scripts into ArcGIS Desktop and ArcGIS Server. With ArcPy, you scripts can access additional ArcGIS functionality to work with your maps and data beyond the geoprocessing framework.

## Course goals

By the end of this course, you will be able to:

- Work with the ArcPy site package in ArcGIS for Desktop and PythonWin.
- Incorporate cursors, use the Describe function, and use Python lists in scripts.
- Automate map document layer and layout management using ArcPy.
- Create script tools and geoprocessing packages.
- Access resources for debugging Python code.
- Understand commonly used ArcPy classes and functions.
- Change table and feature schema for migrating data to new feature classes and tables.
- Create geometry objects for creating and updating features and for input to geoprocessing tools.

## Additional resources

### **ArcGIS Resource Center - <http://resources.arcgis.com>**

This site provides unified access to web-based help, online content, and technical support.

### **Esri GIS Dictionary - [www.esri.com/gisdictionary](http://www.esri.com/gisdictionary)**

This dictionary includes definitions for GIS terms related to geodata, analysis, GIS modeling and web-based GIS, cartography, and Esri software.

## Installing the course data

Some exercises in this workbook require data. Depending on the course format, the data is available on a DVD in the back of a printed workbook or as a data download. To install the data, place the DVD in your disc drive or double-click the data download and follow the instructions in the installation wizard. The data will automatically be installed in the C:\Student folder.



**DISCLAIMER: Some courses use sample scripts or applications that are supplied either on the DVD or on the Internet. These samples are provided "AS IS," without warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or noninfringement. Esri shall not be liable for any damages under any theory of law related to the licensee's use of these samples, even if Esri is advised of the possibility of such damage.**

What to do about winzip/ installation wizard???:

double-click the data download and follow the instructions in the installation wizard. The data will automatically be installed

DISCLAIMER: SOME COURSES USE SAMPLE SCRIPTS OR APPLICATIONS THAT ARE SUPPLIED EITHER ON THE DVD OR ON THE INTERNET. THESE SAMPLES ARE PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. ESRI SHALL NOT BE LIABLE FOR ANY DAMAGES UNDER ANY THEORY OF LAW RELATED TO THE LICENSEE'S USE OF THESE SAMPLES, EVEN IF ESRI IS ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



## Icons used in this workbook



Notes point out additional information, exceptions, or special circumstances that apply to a particular topic or procedure.



Tips provide brief help for performing a task or clarifying concepts.



External resources provide optional, special-interest information about course topics.



Best practices offer industry or professional guidelines, help set goals or priorities, and save time.



Estimated times provide guidance on approximately how many minutes an exercise will take to complete.



Warnings alert you to potential problems or to actions that should be avoided.



# 1

## Running scripts in Python

### Key terms

IDE

Python window

site package

### Introduction

In this lesson, you will explore working with the ArcPy site package and Python to access Python and geoprocessing functionality, automate your workflow processes, and access and update data. You will also learn how to run your scripts in ArcMap.

### Topics covered

- Integrated Development Environments
- The ArcPy site package
- ArcPy functions and classes
- The ArcPy modules
- Running scripts in the Python window

### Learning objectives

After completing this lesson, you will be able to:

- Describe high-level ArcPy site package functionality.
- Use the ArcPy module in scripts.
- Run scripts in the Python window and in PythonWin.
- Find geoprocessing tools and access them in scripts.

## Integrated Development Environment (IDE)

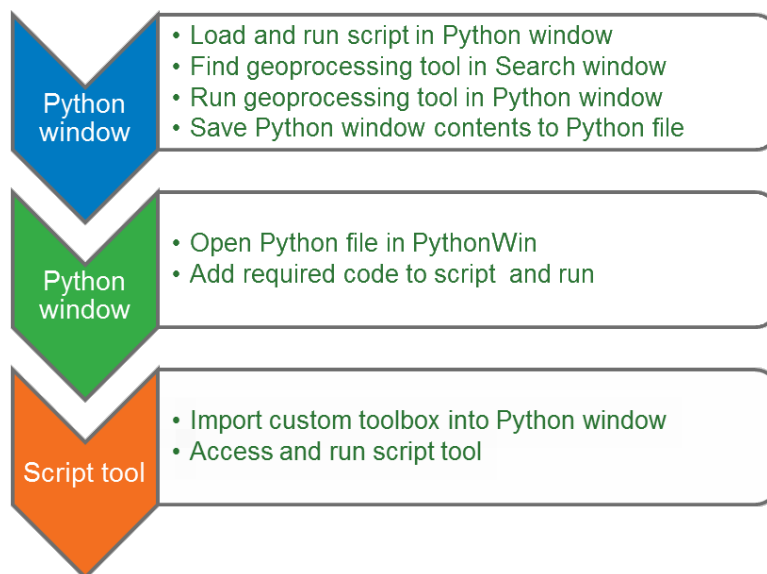
A Python script is simply a text file with a .py file extension. You can use Notepad, WordPad, or other text editors to create and write your Python code. These text editor applications allow you to quickly write your Python code, but there is no way to debug and test run your code with them. The Python command prompt and the Python window are other acceptable place to write your code, but they do not provide any debugging tools to step through your code line by line.

Integrated Development Environments (**IDEs**) provide an environment to write, debug, and run your code from one location. In this course, you will use the PythonWin application as your IDE to develop, debug, and test your scripts.

## Running scripts in Python window

The **Python window** is a convenient location to use geoprocessing tools and Python functionality from within ArcGIS.

**Figure 1.1**  
Workflow for running  
scripts in the Python  
window.



### Why write a script in the Python window?

- It has a simple and intuitive interface that makes learning Python scripting easy—even for new users.
- It provides Python functionality in ArcGIS Desktop.
- Python functions and methods syntax help automatically displays.
- Geoprocessing tool help and arcpy syntax automatically displays.
- Code completion is provided for imported modules, arcpy functions and classes, and tools.

### Why run a Python script in the Python window?

- Quickly test a script before making it into a script tool.
- Test scripting ideas using code snippets before incorporating them into a script.
- Access geoprocessing functionality and view results before incorporating them into a script.
- Combine existing script and test additional functionality, then save as a Python file.



The Python window automatically imports the arcpy module—it is aware of and can work with the layers in the map.



10 minutes



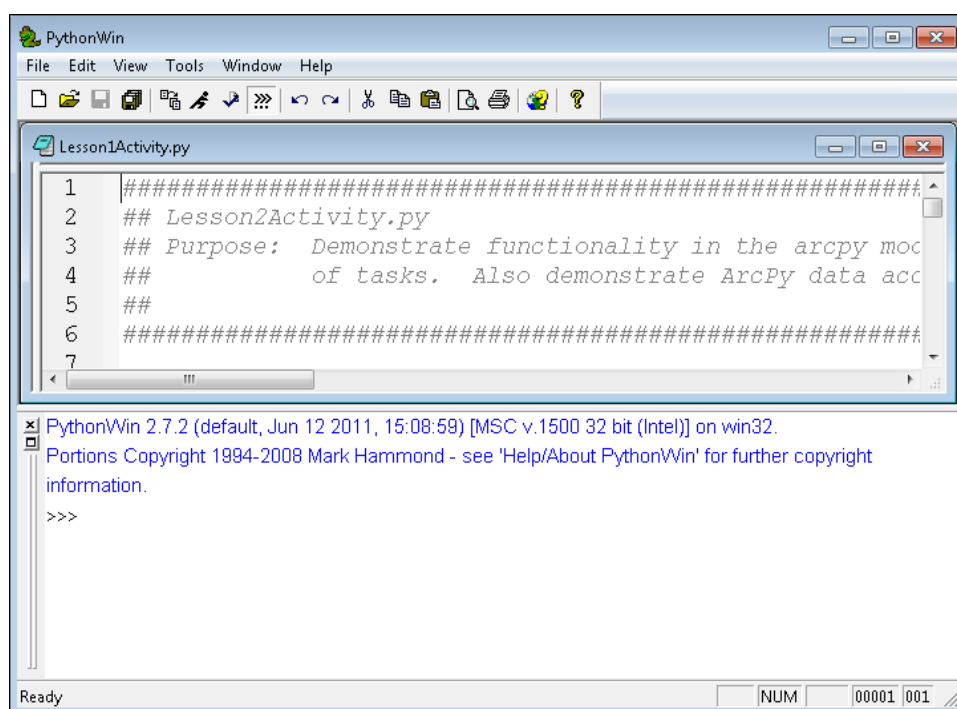
## Exercise 1A: Use the PythonWin IDE (Instructor-led)

There are many ways to develop Python scripts. In this activity, you will explore accessing an existing script in the PythonWin IDE. You will be using the PythonWin IDE for each lesson in the course, where you will work with existing scripts and create new scripts. In this exercise, you will open a script in PythonWin, review the script contents, and then run the script.

In this exercise, you will:

- Become familiar with the PythonWin IDE.

**Figure 1.2**  
The PythonWin IDE.



## Step 1: Open and configure PythonWin

- a From the Taskbar, click the PythonWin shortcut to start the PythonWin IDE.
- b Maximize the PythonWin application.
- c From the File menu, click Open, then navigate to the C:\Student\PYTH\Running\_scripts folder.
- d Click the Lesson1Activity.py script to select it, then click Open.

Now you will configure PythonWin.

- e From the Window menu, click Tile.

This maximizes the script window in PythonWin. Note that the script window docks to the Interactive Window.

- f From the View menu, choose Options. Browse through the tabs in the dialog window and note the settings:
  - **General tab:** In the Interactive Window panel, the check box for Dockable window is selected. This docks the Interactive Window to the bottom of the application.
  - **Editor tab:** In the Margins Width panel, note Line Numbers. Line numbers can be displayed in the Script window by setting this value to 20 or higher.
  - **Format tab:** The font, text size, and text color can be set for the script window.
- g In the PythonWin Options dialog box, click Cancel.

## Step 2: Run the script

- a Read through the script comments.

Notice that the script is using the ArcPy module to access geoprocessing functionality. Import the ArcPy module into the PythonWin Interactive Window to access code complete functionality. You will learn more about ArcPy in this lesson.

- b Click inside the script window.



- c Click the Run icon  on the Standard toolbar, then click OK to run the script.



You could also press the F5 key on your keyboard to run the script.

After a few moments, you receive a message in the Interactive Window, notifying you that the script has completed.

- d In the Interactive Window, scroll up and review the output.

1. What type of information was displayed in the Interactive Window?

---

- e You may choose to run the script again, or, if you are done exploring PythonWin, you may close the application.

## Script workflow and summary

Script workflow:

- Imported the ArcPy module.
- Set the current workspace environment.
- Created a list of feature classes and printed the names.
- Using the same list, ran the GetCount geoprocessing tool for each item.
- Printed the feature class name and the feature count.
- Using a data access cursor, printed the attribute values for each School feature.

Using the ArcPy module and lists of items created by the ArcPy List functions, the script automated a geoprocessing workflow task. Using a list of items, it was easy to automate the process of running the same geoprocessing tool on each item. The script also accessed and displayed data values for a field. The ArcPy data access cursors provided functionality to access and display the data values.

In upcoming lessons, you will learn tools and techniques for fixing syntax errors, debugging your script, and catching errors that occur when the script is run in PythonWin. You will also explore how to create lists and access data values in fields.



ArcGIS 10.1 Help Library:  
*A quick tour of ArcPy*



The third-party site packages, NumPy, Matplotlib, and win32com, are also installed by ArcGIS to provide additional functionality to scripts.

**Figure 1.3**

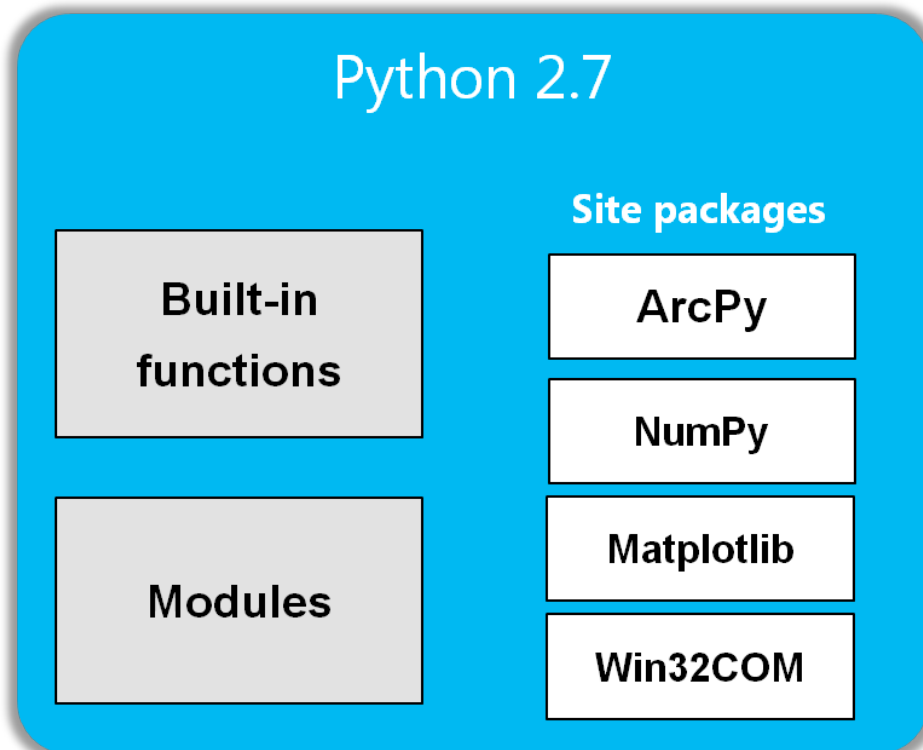
Python 2.7 site packages.

## The ArcPy site package

Geoprocessing functionality has come a long way since it was first introduced. Since the release of ArcGIS 9, geoprocessing functionality can be accessed by running tools in ArcToolbox, creating and running a model in ModelBuilder, running the geoprocessing tools in a Command Line window in ArcMap, or writing a script and running it in an IDE.

ArcPy is a site package that is installed by ArcGIS. (A **site package** is a library that adds functions to Python.) ArcPy adds ArcGIS geoprocessing and map automation functionality to Python. By importing ArcPy into your script, you can perform geographic analysis, data conversion, data management, and map automation workflows with Python. Writing scripts that automate workflow processes using ArcPy functionality can increase efficiency and productivity, and it can also enforce standardized workflows.

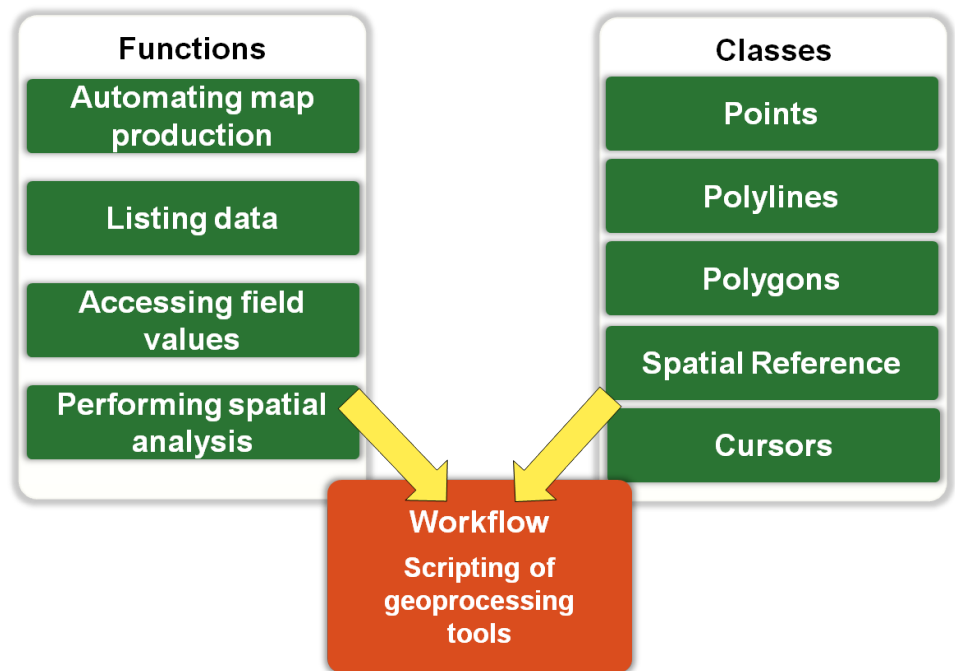
ArcPy is composed of standard functions and classes, and additional modules containing focused functionality for working with data access and map document contents. It also supports the Spatial Analyst and Network Analyst extensions.



## ArcPy functions and classes

When ArcPy is imported into a Python session, the base ArcPy functions and classes and all licensed geoprocessing tools are made accessible to your scripts. In Python, a *function* is a bit of code that performs an operation and typically returns a result. A *class* can be thought of as a template for creating objects.

**Figure 1.4**  
ArcPy base functions and classes.



### ArcPy functions

ArcPy functions support many geoprocessing workflows. For example, with ArcPy functions, you can:

- List data for automating processing of datasets.
- Describe data properties for decision-making in a script.
- Step through individual features in a dataset or rows in a table using the data access objects for reporting, updating field values, or creating new data.
- Manipulate and update map layers in your map document.
- Set and receive parameter values in your script.
- Run geoprocessing tools.
- Add geoprocessor messaging to the progress dialog and results when your script is run in ArcGIS Desktop.
- Manipulate the contents of map documents.
- Publish your map document as a geoprocessing service or package.



ArcGIS 10.1 Help Library:  
*Alphabetical list of ArcPy  
classes*

## ArcPy classes

ArcPy classes can be used to create objects to support ArcPy functions and automate processes to create geometry objects in memory (such as Points, Polylines, and Polygons, as well as SpatialReference, and Extent objects). These objects can be used to represent actual geometry for updating or creating new features, or they can be used in place of actual feature geometry for geoprocessing tools and topological comparisons. These geometry objects can then be used to create, update, and view the geometry in our features.

### The ArcPy modules

In addition to the standard functions and classes, ArcPy contains several modules. Each module contains specialized task-related functions and classes that are accessible in code by specifying ArcPy and the module name, as shown in the following example.

**Figure 1.5**  
ArcPy module import  
syntax example.



**Table 1.1**

**ArcPy modules available in an ArcGIS Desktop install**

ArcPy Module	import/access	Description
Data Access module	arcpy.da	Provides classes and functions for working with data. There is support for converting to/from NumPy arrays and edit sessions along with support for versioning and replication workflows.
Mapping module	arcpy.mapping	Functions and classes for automation of map document (.mxd) management.
Network Analyst module	arcpy.na	Provides helper functions and classes for working with and automating workflows in the Network Analyst extension.
Spatial Analyst module	arcpy.sa*	Provides access to the Spatial Analyst extension functions and operations.
Time module	arcpy.time	Useful classes and functions for working with time deltas and time zones. Can be used in place of the Python datetime module.

\*Nesting several functions and classes together to perform an operation with the arcpy.sa module are explained in detail on the following page.

**Figure 1.6**

Importing the `arcpy.sa` module into a script.



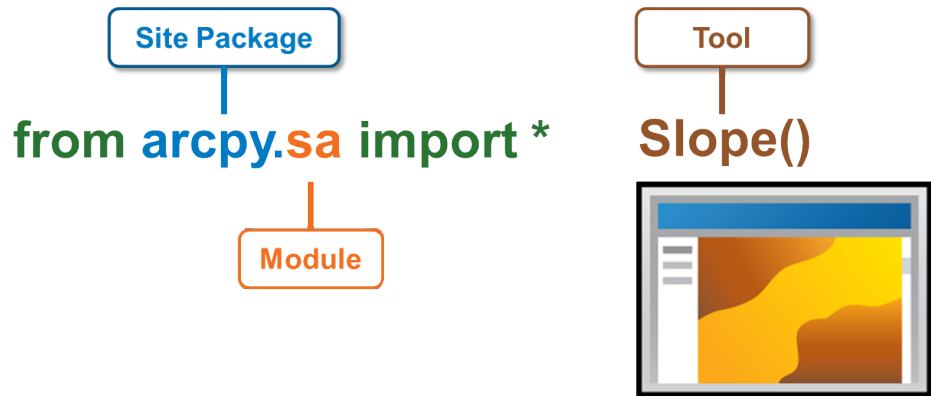
ArcGIS 10.1 Help Library:  
*Importing ArcPy*



ArcGIS 10.1 Help Library:  
*Essential Python add-in concepts.*

## Working with the `arcpy.sa` module

When working with the `arcpy.sa` module functionality, it may be desirable to nest several functions and classes together in single line of code to perform an operation, such as working with the Raster function. The `from-import-*` statement can simplify the code readability, as in the example below:



```
import arcpy
from arcpy.sa import *
arcpy.CheckOutExtension("spatial")
arcpy.env.workspace = "C:/MyData/MyProject.gdb"
ras1 = "Slope("elevation", "DEGREE", 0.3043)
ras2 = "SolarRad"
outRAS = (Raster(ras1) + (Raster(ras2)))
```

## `pythonaddins` module

This module contains functionality to support and work with the new Python add-ins at ArcGIS 10.1. It cannot be used in stand-alone scripts or geoprocessing script tools. Python add-ins at ArcGIS 10.1 will not be covered in this course.

## Choosing a scripting environment

You have options for the environment in which you will access Python functionality and develop your scripts.

The *Python window* is best for writing small bit of codes, running a tool, working out a small workflow, or testing how things work in Python. When writing large amounts of code, you are encouraged to use an *IDE*.

### Benefits to using the PythonWin IDE:

- Allows for the execution of a single line of code.
- Has a Windows look and feel.
- Provides a code syntax checker.
- Allows you to step through the code line by line.
- Allows you to set breakpoints to interrogate variables values.
- Provides code completion in the script window.

### Decision matrix for choosing Python window or PythonWin

Table 1.2

Do you need to...	Python window	PythonWin
Execute a single line of code?	✓	✓
Work directly with layers in ArcMap?	✓	
Test geoprocessing workflows; easily verify results?	✓	
Work with tools to enter, edit, and check syntax in your code?		✓
Work in an Integrated Debugging Environment?		✓

## Tips and best practices

### Common syntax errors

When visually checking your script for syntax errors, first concentrate on these common syntax errors:

- Spelling mistakes; transposition of letters
- Case sensitivity of variable names (e.g., myData is not the same as MyData)
- Path separators—either use two (\\), one (/), or use a lower-case r in front of the path containing a single (\)
- Missing colons on if-else, for-in, and while statements
- Improper indentation (the Check button/TabNanny will catch this type of error)
- Conditional evaluation (==) versus assignment (=)

### Locating geoprocessing tools

Many of the scripts you write may call geoprocessing tools. There are several different methods you can use to locate the desired geoprocessing tool.

- The *Search* window in ArcMap and ArcCatalog can find the tool you are looking for. Simply enter keywords or phrases that describe what the tool does.
- The *Catalog* window provides the Toolbox node in the Catalog tree. Open the node and then open the System Toolboxes to browse through the collection of toolboxes and toolsets.
- The *ArcToolbox* window provides the same browse capabilities as the *Catalog* window.
- The *Search* tab in the *ArcGIS 10.1 for Desktop Help* provides topic keyword search capabilities and list any relevant help topics that use the keyword.



35 minutes



## Exercise 1B: Run scripts in Python

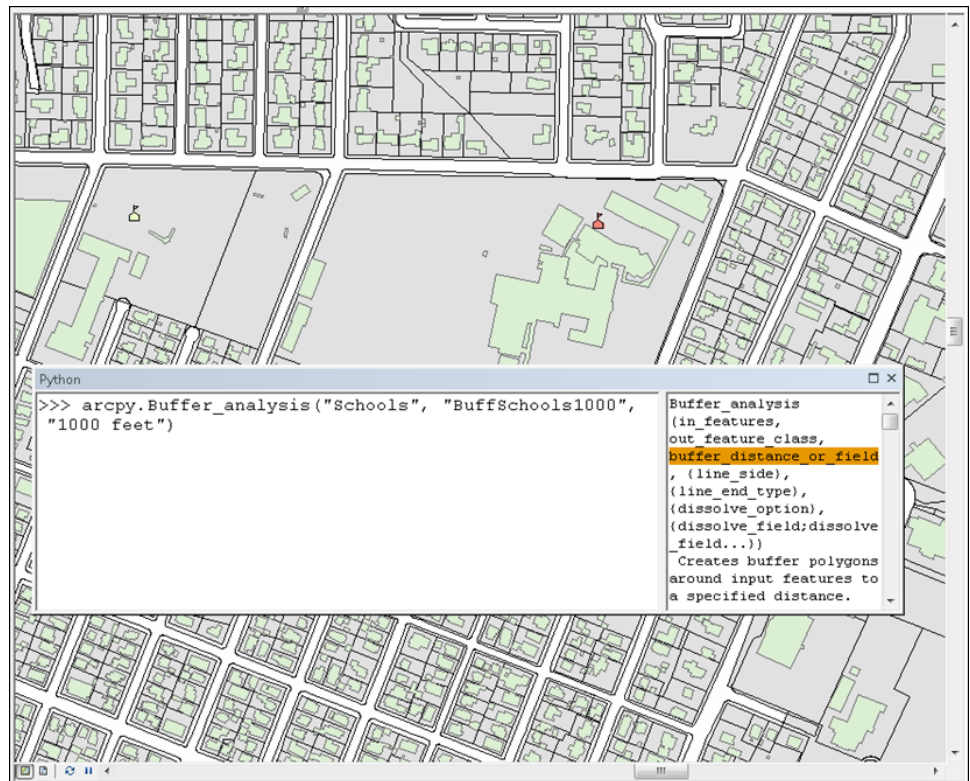
Many geoprocessing tasks can be automated through scripts. Using a Python script, you will automate the process of buffering features a specified distance. You will then add the results to the ArcMap document for display and verification.

In this exercise, you will:

- Run the Buffer tool in the Python window.
- Export Python window code to a Python script.
- Debug and run the script in PythonWin.

**Figure 1.7**


Python window being used in ArcMap to create a buffer in an area of Corvallis.



## Step 1: Buffer schools in Python window

In this step, you will locate and access a geoprocessing tool from the ArcMap Search window, and then run the geoprocessing tool in the Python window. After the tool runs successfully, you will export the Python window contents to a Python file.

Your task is to run the Buffer geoprocessing tool on the Corvallis schools. You will test the geoprocessing workflow in the Python window, verify the results, and then create a Python script.

- a In ArcMap, from the file menu, select Open.
- b Navigate to the C:\Student\PYTH\Running\_scripts folder, and open Corvallis.mxd.
- c Open the Python window and move it to the bottom of the ArcMap display.  
*Hint: From the Standard toolbar, click the Python window button . Or, from the Geoprocessing menu, select Python.*
- d Open the Search window and click Tools.
- e In the Search window, type **Buffer**, then press Enter.

The Search window displays the Analysis toolbox, the Buffer tools, and other matching items.

- f In the Python window, set the current workspace environment setting to the Corvallis geodatabase:  
Type `arcpy.env.workspace = r'C:\Student\PYTH\Running_scripts\Corvallis.gdb'`, then press Enter on your keyboard.
- g From the Search window, drag the Buffer(Analysis) tool into the Python window.

Notice that code complete displays a list of the layers from the table of contents for you to choose from.

- h Double-click "Schools" to enter the `input_features` parameter value.
- i Add a comma to move to the second parameter.

Next you will add the `out_feature_class` parameter value.

- j Type "**BuffSchools1000**", (make sure to include the comma so that you can move to the third parameter).
- k For the `buffer distance` parameter value, type "**1000 feet**" and then enter a close parenthesis.
- l Click just inside the close parenthesis to view the tool syntax.

Verify that your Python window matches the following graphic.

The screenshot shows a Python window with two panes. The left pane contains the following code:

```
>>> arcpy.env.workspace = r'C:\Student\PYTH
\Running_scripts\Corvallis.gdb'
>>> arcpy.Buffer_analysis("Schools", "BuffSchools1000",
"1000 feet")
```

The right pane shows the syntax for the `Buffer_analysis` tool:

```
Buffer_analysis(in_features,
out_feature_class,
buffer_distance_or_field,
{line_side}, {line_end_type},
{dissolve_option},
{dissolve_field;dissolve_fiel
d...})
Creates buffer polygons
around input features to a
specified distance.

INPUTS:
```

When executed, the Buffer tool will return a result that will be added to the table of contents.

- m In the Python window, click outside the close parenthesis, then press the Enter key on your keyboard.

In the table of contents, notice the new layer named BuffSchools1000.

- n Verify that the BuffSchools1000 layer is displaying valid buffer polygons.
- o If the results are not valid: return to the Python window, recall the last command (by pressing Up arrow on the keyboard), change the incorrect parameters, then rerun the tool.

You are now ready to export the Python window commands to a Python script.

- p Right-click in the Python window and select Save As.
- q Navigate to the C:/Student/PYTH/Running\_Scripts folder.
- r For File name, type **MyBufferCorvallisSchools.py**.
- s Click Save.

- t Remove the BuffSchools1000 layer from the table of contents.
- u Close ArcMap. You do not need to save your changes.

## Step 2: Update script in PythonWin

Now you will open your exported script in the PythonWin IDE, add required code, then debug and test the script.

- a In PythonWin, from the C:\Student\PYTH\Running\_scripts folder, open the MyBufferCorvallisSchools.py file.

When a script is run in the PythonWin IDE, it runs outside of the ArcGIS environment. Because of this, you will need to import the ArcPy module into the script.

- b Add a new line at the top of the script.

*Hint:* Click your cursor at the beginning of a line of code, then press the Enter key.

- c Add code at the top of the script to import the ArcPy module:  
Type **import arcpy**.

When debugging and testing your script in a Python IDE, you may need to run the script several times. The `overwriteOutput` environment setting allows the outputs of geoprocessing tools to be overwritten without causing errors. It controls whether tools will automatically overwrite any existing output when run. When set to `True`, tools will execute and will overwrite the output dataset. When set to `False`, existing outputs will not be overwritten, and the tool will return an error.

- d In your script, add a new blank line just above the Buffer line of code.
- e In the blank line, type **arcpy.env.overwriteOutput = True** to set the `overwriteOutput` environment setting to the value of `True`.
- f Add a new line of code at the bottom of the script to print the message **"Script completed"** to the Interactive Window.
- g Check for and correct any syntax errors.

- h Verify that your script matches the script in the following graphic.

```

1 import arcpy
2 arcpy.env.workspace = "C:/Student/PYTH/Running_Scripts/Corvallis.gdb"
3 arcpy.env.overwriteOutput = True
4 arcpy.Buffer_analysis("Schools", "BuffSchools1000", "1000 feet")
5 print "Script completed"
6

```

PythonWin 2.7 (r27:82525, Jul 4 2010, 09:01:59) [MSC v.1500 32 bit (Intel)] on win32.  
 Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further copyright information.  
 >>>

- i To run the script, on the Standard toolbar, click the Run button , then click OK.

The script should complete successfully and print the message "Script completed" to the Interactive Window.

- j Comment out the line of code in your script that sets the `overwriteOutput` setting to True.



To comment out code in your script in PythonWin, you can:

- Type a hash mark at the beginning of the line of code.
- Highlight the line of code, right-click it, then select Source code > Comment out region.
- Highlight the line of code, then press Alt + 3 on your keyboard.

- k Save the script.

Commenting out this line of code and reopening PythonWin will return the script to a default behavior in which attempting to overwrite existing outputs with geoprocessing tools will generate a system error, or exception. In a later lesson, you will learn how to write code to catch and handle errors, so the an exception is not generated when the script is executed.


The final step in the workflow is to verify the script results in the ArcMap application.

- l Close the PythonWin IDE.

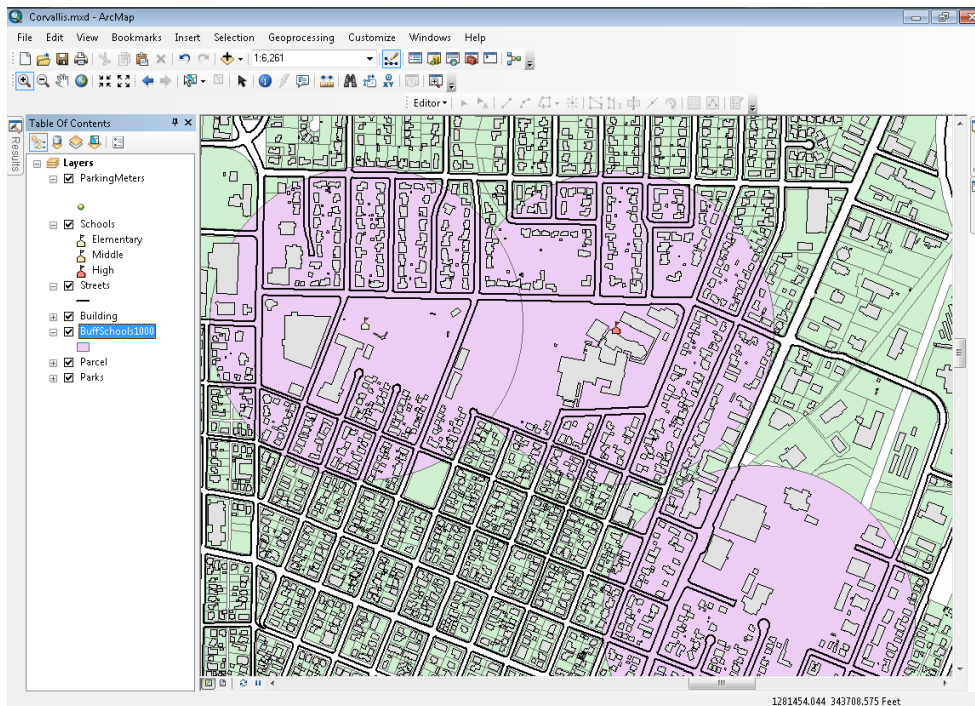
### Step 3: Verify results in ArcMap

Finally, you will verify that the script created the correct buffer polygons around the schools.

- a Open ArcMap, then open the Corvallis.mxd from the C:\Student\PYTH\Running\_scripts folder.
- b Add the C:\Student\PYTH\Running\_scripts\Corvallis.gdb\BuffSchools1000 feature class to the map.

*Hint:* From the Catalog window, either drag the feature class onto the map, or click the Add Data button  to add the feature class.

- c In the table of contents, move the BuffSchools1000 layer so that it displays below the Buildings layer.
- d Verify that the buffer polygons match the buffer polygons in the following graphic.



You have completed the exercise.

- e Close ArcMap.

## Lesson review

1. How can you access geoprocessing functionality in a Python script?

---

---

2. What is ArcPy?

---

---

---

3. In the Python window, how can you view help for a geoprocessing tool?

---

---

---

---

## Answers to Lesson 1 questions

### Exercise 1A: Use the PythonWin IDE (Instructor-led) (page 1-5)

1. What type of information was displayed in the Interactive Window?

**The script displayed information about the geodatabase, the names of the feature classes and feature counts, and field values.**



# 2

## Describing data

### Key terms

Describe function

describe object

### Introduction

The Describe function in the ArcPy site package returns an object that contains various properties on the data element being described. Types of data that can be described include geodatabases, feature classes, tables, rasters, network datasets, topology, geometric networks, shapefiles, toolboxes and tools, folders, coverages, layer files, map documents, and many more.

These describe object properties can be used within a script to report the values, can make conditional decisions that control the flow of the script, and can function as input parameters to geoprocessing tools.

During this lesson, you will look at the describe object properties that are returned for feature classes, tables, rasters, workspaces, and layers.

### Topics covered

- The Describe function
- The generic Describe object
- Describing a feature class
- Describing a raster dataset

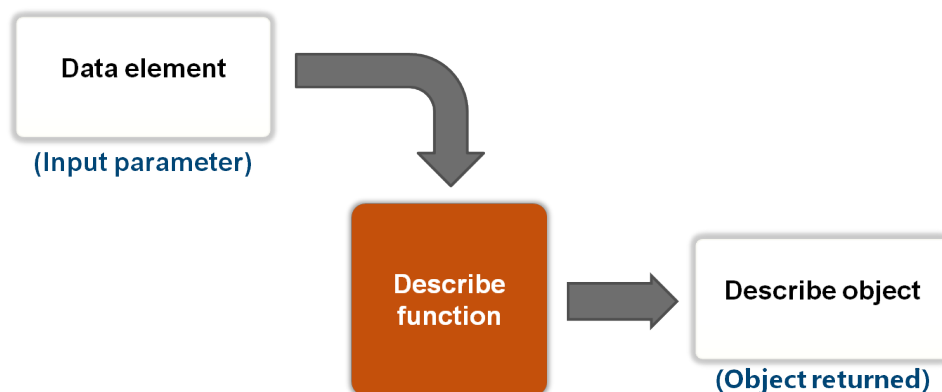
### Learning objectives

- Access data properties using the ArcPy Describe function.
- Use data properties to make conditional decisions in scripts.
- Perform geoprocessing on data using Describe object properties.

## The Describe function

When using the `arcpy.Describe` function, it is important to recognize that the function will always return a **Describe object**. The **Describe function** takes one parameter, which is the data element being described.

**Figure 2.1**  
The `arcpy.Describe` function.



The Describe object has properties that contain descriptive values for the data element being described. For example, if you were describing a workspace, you might be interested in knowing if the workspace is file-based, a local geodatabase, or a remote geodatabase. If you were working with a feature class, you might need to know what type of geometry it can store.

These properties can be used to help you write code that can make decisions based on a property's value. If you were tasked with writing a script that creates a monthly inventory report on the contents of your organization's geodatabases, the Describe object properties could be used to supply the values for the report. The Describe object is dynamic, so the properties returned are for the data element being described. This could include the geodatabase itself, feature classes, tables, rasters, network datasets, and more. The data element being described is not limited to just geodatabase elements.



ArcGIS 10.1 Help Library:  
[Describe \(arcpy\)](#)

## Generic Describe object properties

The Describe function returns a set of generic properties for all Describe objects. These properties include values for the file path, base name, file, name, and data type.

Table 2.1

**Describe object properties**

Property	Data Type	Description
baseName	String	File base name; same as <code>os.path.basename</code> .
catalogPath	String	Full path name.
file	String	File name; same as the baseName in most situations.
name	String	User-assigned name for the describe element.
path	String	File path, same as <code>os.path.dirname</code> .
dataType	String	Data type of the element; FeatureClass for feature classes, ShapeFile for shapefiles, RasterDataset for rasters, Folder for file folder, and Workspace for geodatabases.
extension	String	File extension; gdb for geodatabases and geodatabase elements, shp for shapefiles, blank for folders.
children	List of Describe objects	Python list of items in the geodatabase, such as feature classes and tables.



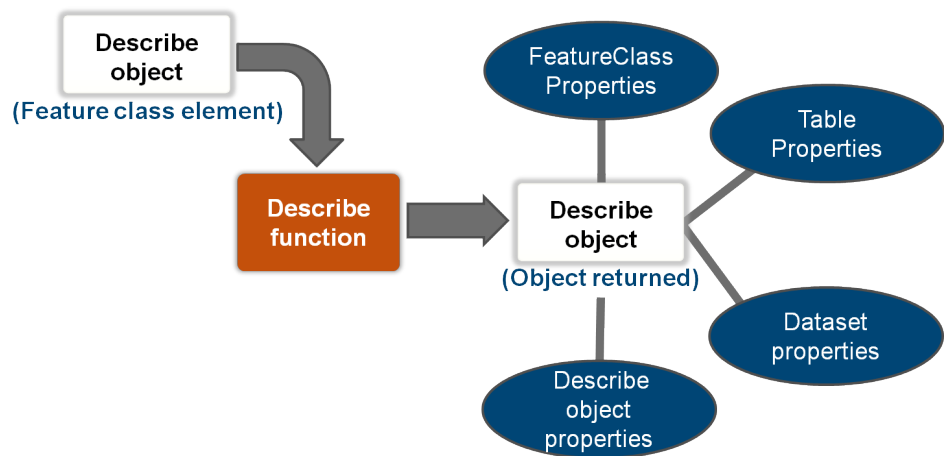
ArcGIS 10.1 Help Library:  
*FeatureClass properties*  
(*arcpy*)

**Figure 2.2**

Process of a feature class element being passed to the Describe function.

## Feature class Describe properties

When a feature class element is passed to the Describe function, the Describe object that is returned contains information about the feature class. Also included in the Describe object is information about the table, the dataset properties, and the generic properties.



**Feature Class properties**

Property	Data type	Description
featureType	String	Information on the feature type being stored in the feature class. Feature classes storing points, lines, and polygons return a value of "Simple".
hasM	Boolean	True only if the geometry supports an m-value.
hasZ	Boolean	True only if the geometry supports an z-value.
hasSpatialIndex	String	True if the feature has a spatial index. All geodatabase feature classes have a spatial index. Shapefiles typically do not.
shapeFieldName	String	The value is typically Shape.
shapeType	String	The geometry shape type. Values are Point, Polygon, Polyline, MultiPoint, and MultiPatch.

Table 2.2

**Table properties**

Property	Data type	Description
hasOID	Boolean	True if the table has an ObjectID field.
OIDFieldName	String	The name of the ObjectID field, if it exists. External tables will not have an ObjectID field.
fields	List of field object	A list of Field objects for the table. Each field object in the list will have properties about the field.
indexes	List of index objects	A list of Index objects for the table.

Table 2.3

**Dataset properties**

Property	Data type	Description
canVersion	Boolean	For remote datasets, a True value indicates that the dataset can be versioned.
datasetType	String	Indicates the type of dataset that is being described. Some possible values: FeatureDataset, FeatureClass, RasterDataset, Terrain, Locator, and Table (this is not an exclusive list).
DSID	Integer	The ID of the dataset; a numeric value.
extent	Extent	An extent object whose properties provide the X & Y of the lower-left and upper-right corners of a rectangle bounding the features in the dataset. There are several spatial comparison methods also included in the extent object (such as contains, crosses, overlaps, touches).
isVersioned	Boolean	For remote datasets, True indicates the dataset is versioned.
MExtent	String	If the dataset is m-value enabled, it will contain the MMin and MMax values.
spatialReference	SpatialReference	Returns a spatialReference object for the dataset. The spatialReference object will contain properties describing the coordinate system of the dataset.
ZExtent	String	If the dataset is z-value enabled, it will contain the ZMin and ZMax values.

Table 2.4



ArcGIS 10.1 Help Library:  
Raster Dataset properties  
(arcpy)

## Raster Describe properties

When a raster dataset element is passed to the Describe function, the Describe object returned contains information about the raster dataset. Also included in the Describe object is information about the raster band, the table, the generic properties, and the dataset properties.

**Figure 2.3**  
Process of a raster dataset  
element being passed to  
the Describe function.

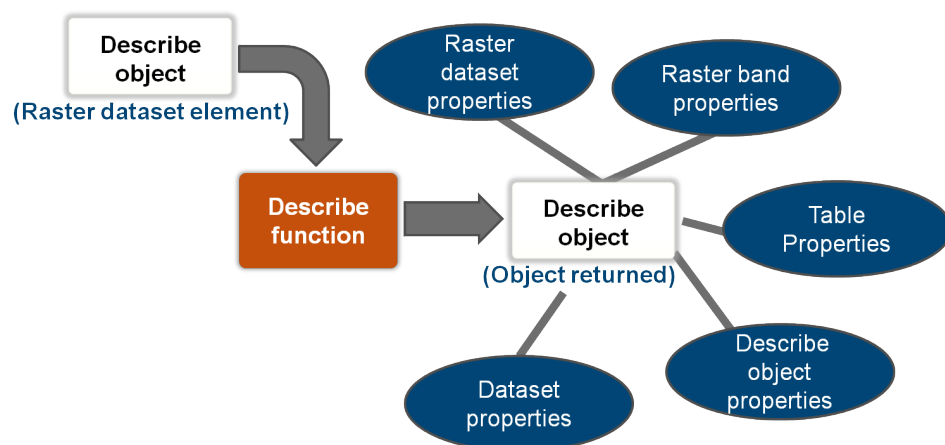


Table 2.5

**Commonly accessed raster dataset properties**

Property	Data Type	Description
bandCount	Integer	The number of bands (sensors) in the raster dataset.
compressionType	String	The raster compression type. Possible values are LZ77, JPEG, JPEG2000, and None.
format	String	The raster dataset format. Possible values are Grid, ERDAS IMAGINE, or TIFF.

Table 2.6

**Describe object properties for a raster band**

Property		Description
height	Integer	Number of rows in the raster band.
width	Integer	Number of columns in the raster band.
pixelType	String	Raster band pixel type. Values indicate signed and unsigned pixel types from 1 bit through 32 bit, floating point single, and double precision pixel types.
isInteger	Boolean	True if the raster band has integer values.



## Describing a feature class and raster

### Tips for using the Describe object

You can use the Describe object `dataType` property to determine what is being described. Some of the values returned by the `dataType` property include:

- FeatureLayer
- FeatureClass
- File
- Folder
- LasDataset
- Layer
- MosaicDataset
- NetworkDataset
- RasterBand
- RasterDataset
- ShapeFile
- Table
- TextFile
- Toolbox

The Describe object contains properties that are dynamic. This means that when different data types are described, different describe properties will be available. Because of the dynamic nature of the Describe object, no drop-down of properties is provided.

If you try to access a property that the Describe object does not have, you will receive either an error or an empty value. To check if a particular property is contained in the object returned from the Describe function, you can use the Python `hasattr()` function in your script .

The `hasattr()` function returns a Boolean value. An example of using the function with the Describe object is shown below.

```
import arcpy
desc = arcpy.Describe("C:/Data/Mydata.gdb/Topo")
if hasattr(desc, "name"):
    print "Name: " + desc.name
if hasattr(desc, "bandCount"):
    print "Raster bands: " + desc.bandCount
if hasattr(desc, "shapeType"):
    print "Shape type: " + desc.shapeType
```

## Fill-in-the-blank

Provide the missing information needed to complete the script.

### Describe a feature class

```
# This script describes the Climate
# feature class in the SanDiego geodatabase.
# Import the ArcPy site package and
# set the workspace environment
import arcpy
arcpy.env.workspace =
"C:/Student/PYTH/Describing_data/SanDiego.gdb"
# Obtain describe properties of Climate
# feature class
desc = arcpy.Describe("Climate")
# Print the geometry type to the Interactive Window
print "Geometry type: " + desc._____
if desc._____ == "Simple":
    print "Climate contains simple features"
else:
    print "Climate contains non-simple features"
```

### Describe a table

```
# This script describes a table
# from the World.gdb.
# Import the ArcPy site package and
# set the workspace environment
import arcpy
arcpy.env.workspace =
"C:\Student\PYTH\Describing_data\World.gdb"
# Describe the Lakes feature class
desc = arcpy.Describe("Lakes")
# Display the field name and data type
# for each field
fldList = desc._____
for _____ in fldList:
    print "{0} : {1}".format(
        fld._____, fld._____)

```

## Describe a raster

```
# This script describes a raster from
# the Tahoe\Emer folder.
# Import the ArcPy site package and
# set the workspace environment
import arcpy
arcpy.env.workspace =
"C:\Student\PYTH\Describing_data\Tahoe"
desc = arcpy.Describe("Emer\Erelev")
# Print dataType, should be "RasterDataset"
print "Describe Data type is: " + desc.dataType
# Print number of bands, format and compression
print "Band count: {0}, format: {1}, " + \
      "compression: {2}".format(
      desc._____, desc._____, desc._____)
# Print spatial reference name
print desc._____.name
```



50 minutes



## Exercise 2: Work with the Describe object

The city of Corvallis, Oregon has requested that you write a script to report the contents of their geodatabase. They are interested in reporting the name, geometry type, Spatial Reference name, and number of features for each feature class. You will write a script to report the requested information using the Describe function.

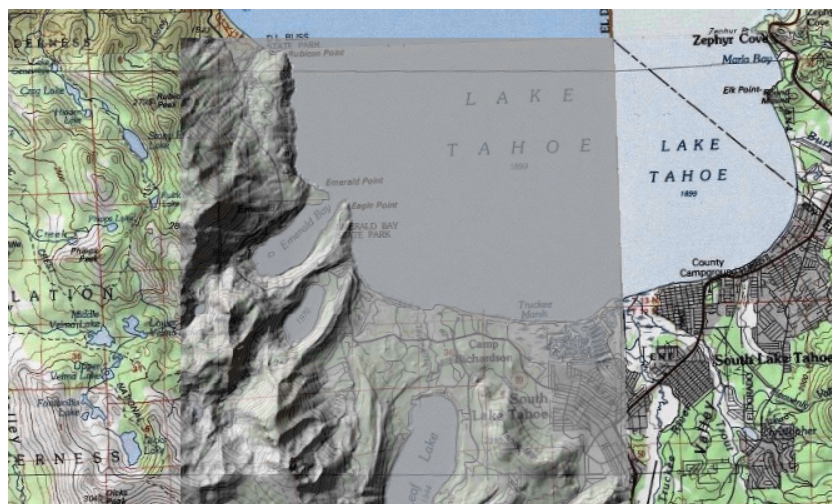
You have decided to go hiking in the Lake Tahoe area and would like to make some maps of the area. The GIS director for El Dorado county has provided you with a data set of elevation, hillshade, lake, land cover, and ownership rasters covering the Lake Tahoe area. After viewing the data, you have narrowed your area of interest to the area covered by the erelev raster. You will write a script to clip the rasters datasets for Lake Tahoe using the extent object of the erelev raster.

In this exercise, you will:

- Describe a feature class.
- Describe a geodatabase.
- Automate describing feature classes contained in a geodatabase.
- Use the Describe object properties with a geoprocessing tool to clip rasters.

**Figure 2.4**

Results of executing code to clip rasters by an extent object.



## Step 1: Describe a feature class and a geodatabase

In this step, you will create a new script and write code to describe the Schools feature class in the Corvallis geodatabase. The results of describing the feature class will be printed to the Interactive Window.

- a In PythonWin, from the File menu, select New.
- b In the New dialog, box, select Python Script, then click OK.
- c Press **Alt+B** to add a header to your script. Add information inside the header that describes your script.
- d Import the ArcPy module and set the workspace to the `C:\Student\PYTH\Describing_data\Corvallis.gdb` geodatabase.
- e In PythonWin, save your new script as **MyDescribe\_schools.py** to the `C:\Student\PYTH\Describing_data` folder.
- f In the Interactive window, type **import arcpy**, then press Enter on your keyboard.



This will provide you with code complete behavior in the script window.

Next, you will work with the object returned from the `Describe` function.

- g Enter the following line of code in your script to describe the Schools feature class:  

```
desc = arcpy.Describe('Schools')
```

The `desc` variable contains the `Describe` object returned from the `arcpy.Describe` function.

- h Add code to report on the `Describe` object values, using a **print** statement and the `str.format()` function:

```
print "Name: {0} Shape: {1} Type: {2}".format(desc.name,
      desc.shapeType, desc.datasetType)
```


- i Add a **for-in** loop to the script to print out the Python list of field objects names:

```
for fld in desc.fields:
    print "\t" + fld.name
```



You can use escape sequences in strings to control the output from the print function. A `'\t'` will insert a horizontal tab in the string, a `'\n'` will insert a line feed in the string. A table listing escape sequences is detailed in the Python v2.7 documentation in *Section 2.4.1 - String literals*.

Now you will check your script for syntax errors.

- j** On the Standard toolbar, click the Check button .

Next, you will write code to describe the Corvallis geodatabase. The City needs information on the geodatabase type, the release number, and the file path.

- k** Describe the Corvallis geodatabase. Assign the Describe object to a variable named **descGDB**.
- l** Using a **print** statement, report the workspace type, release number, and file path.

Verify that your script code for describing the Corvallis geodatabase looks like the following code:

```
# Describe the Corvallis geodatabase.
descGDB = arcpy.Describe(arcpy.env.workspace)

# Report on the workspace type, release number and file path
print "GDB Type: {0} Release: {1} Path: {2}".format(descGDB.workspaceType,
                                                    descGDB.release,
                                                    descGDB.path)
```

- m** Add code to the script that will inform the end-user that the script has completed.
- n** Check your script for syntax errors, then run your script.
- o** Upon successful execution of your script, close the script in PythonWin, but do not close PythonWin.

## Step 2: Describe a list of feature classes

In this step, you will create a new script and write code to describe the feature classes in the Corvallis geodatabase. The results of describing the feature classes will be printed to the Interactive Window.

- a** Create a new script in PythonWin, and add a header to your script.

*Hint:* Press **Alt+B** on your keyboard to add a header.

- b** Add information inside the header describing your script.
- c** Save your new script as **MyDescribe\_FeatureClasses.py** in the C:\Student\PYTH\Describing\_data folder.
- d** Add code to import the ArcPy module, then set the workspace to the **C:\Student\PYTH\Describing\_data\Corvallis.gdb** geodatabase.

For the next part of this script, you will work with a Python list of feature class names. The `arcpy.ListFeatureClasses()` function will be used to create the list.



ArcPy List functions will be explored in the next lesson.

- e** Enter the following line of code into your script to create the list of feature class names:  
**`fc_list = arcpy.ListFeatureClasses()`**

A `for-in` loop will be used to iterate through the list of feature class names returned from the `arcpy.ListFeatureClasses()` function. Inside the `for-in` loop, you will pass each name to the Describe function.

- f** Enter the following lines of code to iterate through `fc_list`:  
**`for name in fc_list:`**  
**`desc = arcpy.Describe(name)`**
- g** In the ArcGIS 10.1 Help Library, search for the **Get Count (Data Management)** geoprocessing tool.

*Hint:* In ArcMap, use the Help menu or the Search window; or, in a browser, go to the ArcGIS Resource Center ([www.resources.arcgis.com](http://www.resources.arcgis.com)).

1. What is the syntax for this tool?

---

- h** Within the `for-in` loop, add a line of code to the script that assigns the return object from the **`arcpy.GetCount_management()`** function to a variable named **`featCount`**. Be sure to pass the name variable to the `in_rows` parameter for the tool.

Verify that your `for-in` code looks like the following:



```
for name in fc_list:
    desc = arcpy.Describe(name)
    featCount = arcpy.GetCount_management(name)
```

For the final part of the script, you will write code to print the following:

- Name of the feature class
- Shape type supported by the feature class
- Spatial Reference name
- Number of rows in the feature class


- i** Within the `for-in` loop, add the following line of code to the script to print the variable values using the `str.format()` function:

```
for name in fc_list:
    desc = arcpy.Describe(name)
    featCount = arcpy.GetCount_management(name)
    print "Name: {0} Shape: {1} SR: {2} Count: {3}".format(
        name, desc.shapeType, desc.spatialReference.name, featCount)
```

- j** Add a new line of code at the bottom of the script to print a message indicating that the script is complete.

- k** Check the code syntax and run the script.



In addition to checking for syntax errors, the Check button  saves your script.

- l** Once your script runs successfully, close it and leave PythonWin open.

In your script, you wrote code to work with a list of feature classes. For each feature class in the list, you described the feature class and determined the number of features using a geoprocessing tool. You then reported on the feature class describe properties and the feature count for each feature class.

In the next step, you will write a script that uses Describe object properties as input to a geoprocessing tool.

### Step 3: Clip raster datasets with Describe object properties

In this step, you will work with Describe object properties to clip raster datasets.

In preparation for your planned hike in the Lake Tahoe area, you will work with the set of rasters provided by the county. Your task is to write a script that will clip the rasters to the extent of the `erelev` raster. The clipped rasters will be used later to produce maps of the hiking area.

- a In the ArcGIS 10.1 Help Library, search for a geoprocessing tool that can clip raster datasets to create a spatial subset of the input raster. When viewing the results, exclude from consideration the geoprocessing tools that require the Spatial Analyst extension.

2. What geoprocessing tool can you use to clip rasters?

---

3. What are the required parameters for the tool?

---

4. For the rectangle parameter of the tool, what are the four coordinates that need to be specified (list them in order)?

---

- b In PythonWin, create a new script and add a header.
- c In your new script, import the ArcPy module and specify **C:/Student/PYTH/Describing\_data/Tahoe/All** as the current workspace.
- d Name your script **MyClip\_rasters.py** and save it to the C:/Student/PYTH/Describing\_data folder.

You will use the C:/Student/PYTH/Describing\_data/Tahoe/Emer/erelev raster as the clipping rectangle extent.

- e Add code to call the **arcpy.Describe()** function on the **C:/Student/PYTH/Describing\_data/Tahoe/Emer/erelev** raster. Assign the Describe object to a variable named **desc**.
- f Add a line of code to assign the **extent** object from the Describe object to a variable named **rasExtent**.

The extent object will provide the rectangle parameter for the Clip tool.

- g Enter the following line of code in your script to obtain a list of rasters in the current workspace:  
**ras\_list = arcpy.ListRasters()**

A `for-in` loop will be used to iterate through the list of raster datasets returned from the `arcpy.ListRasters()` function. Inside the `for-in` loop, you will pass each raster name to the geoprocessing tool that clips the rasters.

- h Enter the following line of code to create the `for-in` loop:  

```
for name in ras_list:
```
- i Inside the `for-in` loop, add code to call the Clip (Data Management) geoprocessing tool.

There are three required parameters for the tool.

- j Using your answers to the questions earlier in this step, pass the three required parameters to the `arcpy.Clip_management` tool. For the rectangle parameter, use the string equivalent of the object to pass the four required values (as in `str(rasExtent)`). For the output raster name, concatenate `_clip` to the name variable (as in `name + '_clip'`).

- k Verify that your `for-in` loop code looks like this:

```
for name in ras_list:
    arcpy.Clip_management(name, str(rasExtent), name + '_clip')
```

- l For the final line of code in the script, add a `print` statement to inform the end-user the script is completed.

Your completed script should look like the following:

```
# Import the ArcPy module and set the current workspace
import arcpy
arcpy.env.workspace = "C:/Student/PYTH/Describing_data/Tahoe/All"

# Describe the erelev raster and assign the extent object to a variable
desc = arcpy.Describe("C:/Student/PYTH/Describing_data/Tahoe/Emer/erelev")
rasExtent = desc.extent

# Obtain a list of rasters in the workspace
ras_list = arcpy.ListRasters()

# Loop through the list and use the Clip_management tool.
for name in ras_list:
    arcpy.Clip_management(name, str(rasExtent), name + '_clip')

print "Script completed"
```

- m Debug and run your script.

It may take a few moments to process the rasters.

- n In either ArcCatalog or the ArcMap Catalog window, confirm that the clipped rasters were created in the C:\Student\PYTH\Describing\_data\Tahoe\All workspace.



You may need to create a new folder connection to the C:\Student\PYTH folder in your Catalog window.

5. Why were the rasters clipped in the Tahoe\All folder?

---

- o If you are not going to complete the Challenge step, close all open applications.

## Challenge: Describe dataset and coordinate system properties



15 minutes



The time to complete the Challenge step is not included in the estimated exercise time of 50 minutes.

Using datasets of your choice, or data you have brought with you, write a script to describe the following:

- All of the feature classes in your data folder or geodatabase. Report on the Describe object properties of your choice.
- A dataset of choice (either a feature class or feature dataset). Report on the name of the spatial reference (`spatialReference.name`) and the extent of the dataset.

## Lesson review

1. When describing a feature class, what properties can be useful in a script?

---

---

---

---

---

---

---

---

2. What is returned by the extent property of a dataset?

---

---

3. How can you obtain the name of a spatialReference for a feature class?

---

---

---

## Answers to Lesson 2 questions

### Exercise 2: Work with the Describe object (page 2-13)

1. What is the syntax for this tool?

**`arcpy.GetCount_management(in_rows)`**

2. What geoprocessing tool can you use to clip rasters?

**The `arcpy.Clip_management()` geoprocessing tool.**

3. What are the required parameters for the tool?

**`in_raster`, `rectangle`, `out_raster`**

4. For the rectangle parameter of the tool, what are the four coordinates that need to be specified (list them in order)?

**X-Minimum, Y-Minimum, X-Maximum, Y-Maximum**

5. Why were the rasters clipped in the Tahoe\All folder?

**The `arcpy.Clip_management` tool works with the `arcpy.env.workspace` setting if an explicit path is not set for the output raster.**

## Challenge solution: Describe dataset and coordinate system properties

The challenge code should include code for looping through all the feature classes in a folder or geodatabase. Values from the Describe object returned should be printed to the Interactive Window.

The challenge code should also include code for returning the Spatial Reference name and the dataset extent values to the Interactive Window.



# 3

## Automating scripts with lists

### Key terms

iterate

### Introduction

One of the primary tasks in scripting is automating the processing of data with a list of datasets. The ArcPy site package has many List functions that are built to return lists for different types of data. In this lesson, you will explore these ArcPy List functions.

### Topics covered

- The ArcPy List functions
- Creating lists
- Iterating through lists

### Learning objectives

- Determine the proper List function to use for a task.
- Automate a workflow to perform geoprocessing using lists.

## The ArcPy List functions

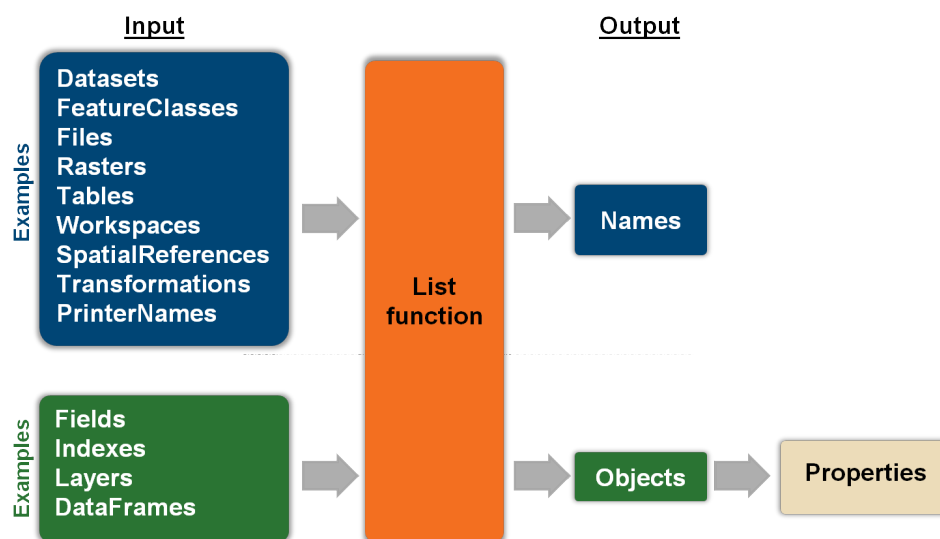
*What are some advantages to working with lists of data?*

The ArcPy site package provides a number of List functions that return a list of values. These List functions can provide you with a list of feature class names in a geodatabase, shapefile names in a system folder, table names in a geodatabase, field names in a table or feature class, and many other lists.

The scripts that you write can **iterate** through the list and perform defined tasks. This means that the task will be performed on each item in the list. For example, you could run a geoprocessing tool on an item or simply report the name to the application.

**Figure 3.1**

The ArcPy List functions.



The ArcPy List functions can be grouped by what is being listed. There are list functions for listing data, spatial reference names and valid transformations, product installations and available printers, and lists of fields and indexes.

The `arcpy.ListFields()` function is an example of a List function returning a list of objects. Each item in the returned list is a field object that contains properties of the field, such as name, type, length, required.

Table 3.1

**The ArcPy List functions for listing data**

List function	What is returned?
ListFields	List of field objects for the specified dataset
ListIndexes	List of attribute index objects for the specified dataset
ListDatasets	List of dataset names in the current workspace
ListFeatureClasses	List of feature class names in the current workspace
ListFiles	List of file names in the current workspace
ListRasters	List of rasters in the current workspace
ListTables	List of table names in the current workspace
ListWorkspaces	List of workspaces in the current workspace
ListVersions	List of versions the connected user has permissions to use
ListInstallations	List of the installation types on the computer
ListPrinterNames	List of printer names available in the session
ListSpatialReferences	List of available spatial reference names
ListTransformations	List of valid transformations names for projecting from one coordinate system to another
ListUsers	List of named tuples containing information on users who are connected to an enterprise geodatabase



ArcGIS 10.1 Help Library:  
*Listing data*

Many of the listed functions accept a wildcard for filtering names, and require that the current workspace is set in the environment settings before using the function. The wildcard parameter defines a name filter for the function, so only the names that pass through the filter are stored in the list created by the function. Keywords can also be provided to restrict the supported types for the function to a specific type. If not specified in the function's parameter, or if the keyword 'All' is specified, all supported types for the function are returned to the list.

To work with your list, use a `for-in` loop to iterate through the list one item at a time. The list can also be manipulated with standard list methods such as `sort`, `append` and `reverse`.



You can also use the search field to find the topic you are looking for.

## Explore the ArcPy List functions

In this activity, you will use the ArcGIS 10.1 for Desktop Help to answer questions about the ArcPy List functions.

There are two ways to access the ArcGIS 10.1 Desktop Help Library:

- In ArcMap, from the Help menu, select ArcGIS Desktop Help.
- In a browser, open [www.resources.arcgis.com](http://www.resources.arcgis.com):
  - At the top of the webpage, click Help.
  - In the ArcGIS 10.1 Help, click Desktop.
  - Click Geoprocessing.

In the ArcGIS 10.1 for Desktop Help Library, navigate to

- *Geoprocessing*
- *ArcPy*
- *ArcPy functions*
- *Listing data* topics

Each ArcPy List function will have its own help topic. Use the information provided to answer the following questions:

1. The `ListFeatureClasses()` function returns a list of items. What data type is returned from the function? What do the values contain?

---

2. The `ListFields()` function returns a list of items containing field objects. What are some of the field object properties that you can access?

---



---

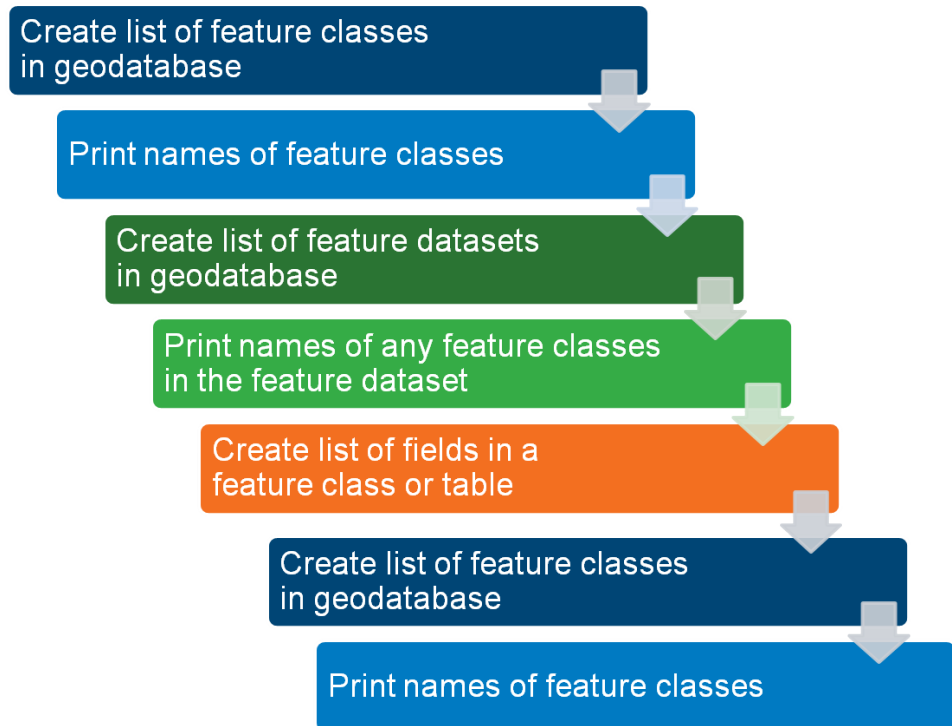
3. Provide a list of ArcPy List functions that must have the current workspace environment set before the function can be used.

---

## Working with List functions

Workflow to automate your tasks using ArcPy List functions:

**Figure 3.2**  
Workflow to automate  
tasks using ArcPy list  
functions.



## List data

Using the script comments, fill in the code that is missing in the three incomplete scripts. Use the workbook and Desktop Help resources to complete the activity.

### List datasets:

```
# This script creates a Python list of feature
# datasets from the World.gdb geodatabase.
# A for-in loop is used to iterate through the
# list and print the dataset name(s)
# to the Interactive Window
import arcpy
arcpy.env.workspace =
    "C:/Student/PYTH/Automating_scripts/World.gdb"
# Fill in the blanks
dataset_list = arcpy._____()
for dataset in _____:
    print _____
```

### List field names and type:

```
# This script creates a list of field names
# from the MajorAttractions feature class.
# A for-in loop is used to iterate through the
# list of field objects and print
# each field name and type to the Interactive Window
import arcpy
arcpy.env.workspace =
    "C:/Student/PYTH/Automating_scripts/SanDiego.gdb"
# Fill in the blanks
field_list = arcpy.ListFields("_____")
for field in field_list:
    print _____._____
    print _____._____
```

### List rasters and build pyramids

```
# This script creates a list of rasters
# in the Tahoe\Emer folder that start with 'e'.
# For each raster name in the list, run
# the BuildPyramids_management tool.
import arcpy
arcpy.env.workspace =
    "C:/Student/PYTH/Automating_scripts/Tahoe/Emer"
# Fill in the blanks
raster_list= arcpy._____("____")
for raster in raster_list:
    arcpy.BuildPyramids_management(_____)
```





40 minutes



## Exercise 3: Automate scripts with the ArcPy List functions

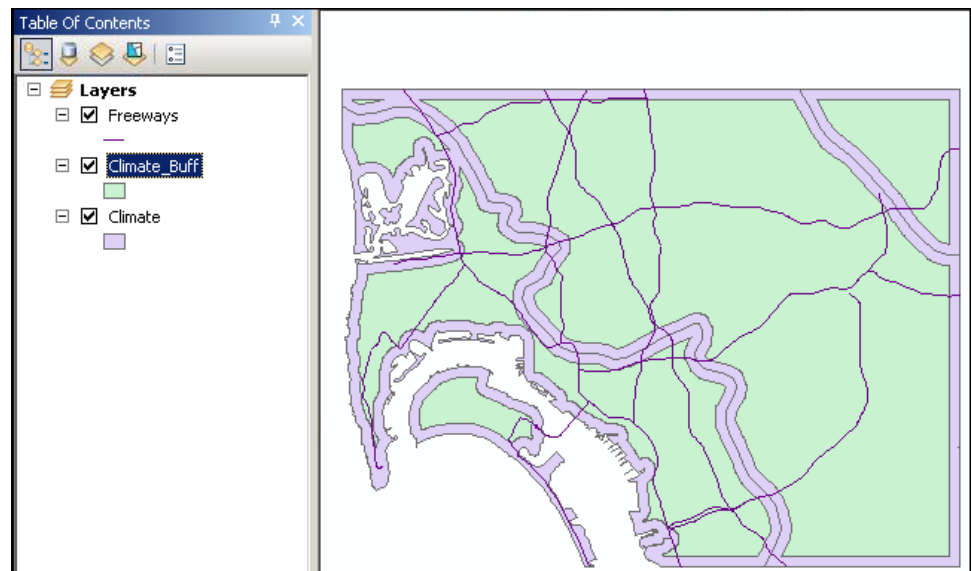
The GIS department of San Diego regularly performs documentation tasks on the contents of their geodatabase. They also perform proximity analysis on the features in the geodatabase. As the Python specialist in the GIS department, you have been asked to create two scripts. One script will document the geodatabase feature classes. The other script will automate the preparation of the feature classes for the proximity analysis.

In this exercise, you will:

- Document the field properties in a feature class and then output and view the results in a text file.
- Use the Buffer\_analysis tool on the geodatabase feature classes (buffer distance will be determined by shapeType).
- Verify script results in ArcMap.

**Figure 3.3**

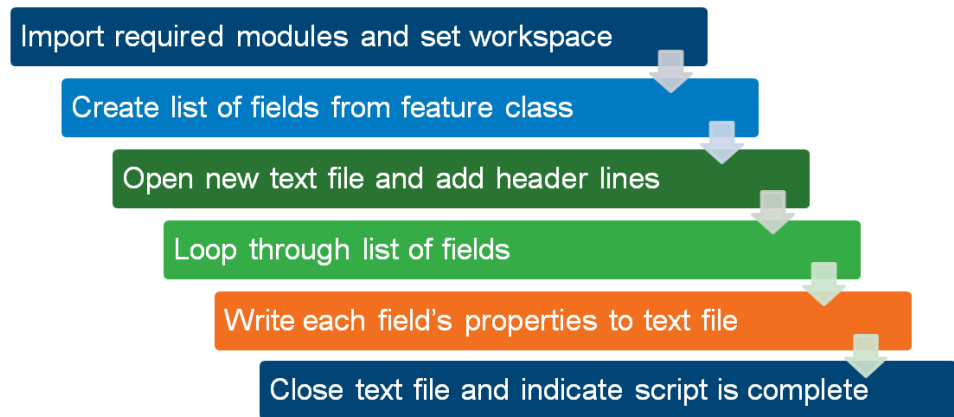
Results of the created buffer featured classes displayed in the map.



**Figure 3.4**

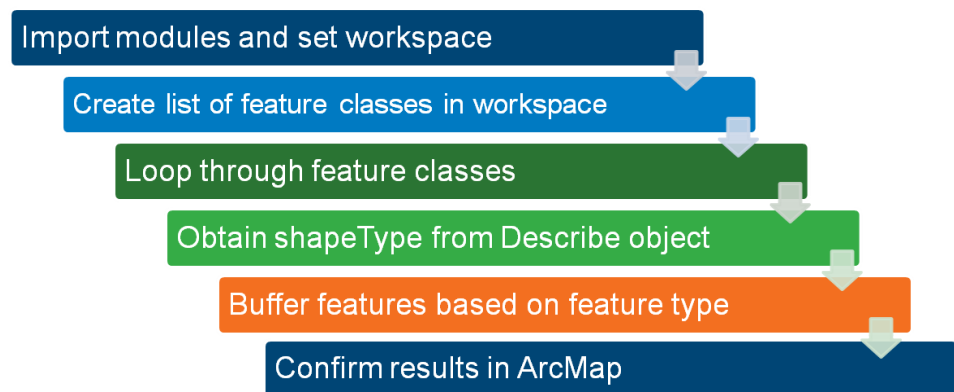
Workflow to document field properties for a table or feature class.

### Workflow for first script:

**Figure 3.5**

Workflow for automating a geoprocessing task on a list of feature classes.

### Workflow for second script:



## Step 1: List field properties

In this step, you will create a new script that lists all of the field names, field types, field lengths and field alias name properties. The output from the script will be written to a text file.

The detailed workflow for the script is:

- Import required modules and set the workspace environment.
  - Create list of fields from feature class.
  - Open a new text file in write mode and add header lines.
  - Loop through the list of fields.
  - Write each field's properties to the text file.
  - Close the text file and inform the user the script is completed.
- a In Python Win, create a new script named **MyListFieldProperties.py** and save it to the C:/Student/PYTH/Automating\_scripts folder.
  - b Add a header to the script and document the purpose of the script.
  - c In your script, import the ArcPy module and the os module.
  - d Assign the value of "C:/Student/PYTH/Automating\_scripts" to a variable named **wksp**.
  - e Set the current workspace environment to:  
`arcpy.env.workspace = os.path.join(wksp, "SanDiego.gdb")`.



The `os.path.join()` function joins two items together intelligently, with a directory separator between the two items.

The first task is to create a list of fields from a feature class.

- f In the ArcGIS 10.1 Help Library, search for the *Listing data* topic. The help topic displays the ArcPy functions that create lists for cataloging data.

1. Which of these functions returns a list of fields?

---

- g** Using the answer, write code using the ArcPy List function to create a list of field names on the **MajorAttractions** feature class. Assign the list returned from the function to a variable named **field\_list**:

```
field_list = arcpy.ListFields("MajorAttractions")
```

Next, you will write code in the script to open a text file for write access, and then add a header to the file contents.

- h** Add the following code to the script to open the text file and write two header lines:

```
txtFile = open(os.path.join(wksp, "MajorAttractions.txt"), "w")  
txtFile.write("MajorAttractions field information" + "\n")  
txtFile.write("-----" + "\n")
```



You should always concatenate a newline character ('\n') to the end of the string value before passing the string to the `file.write()` method. The newline character will indicate the end of the line.

The next task in the script is to write each field's name, type, and length properties to the text file.

- i** Create a **for-in** loop on **field\_list**, with the item value being passed to a variable named **field**.
- j** Within the loop, use the **str.format()** function to assemble a string value containing the **Name**, **Type**, and **Length** properties of the **field** object. Assign it to a variable named **line**:

```
line = "Name: {0}, Type: {1}, Length: {2}\n".format(  
field.name, field.type, field.length)
```

- k** Add code to pass the **line** variable as the parameter to the **write()** method on **txtFile**:
- ```
txtFile.write(line)
```

- l** Press the Enter key twice to dedent out of the loop.

- m** Verify that the **for-in** code looks like the following:

```
for field in field_list:  
    line = "Name: {0}, Type: {1}, Length: {2}\n".format(  
        field.name, field.type, field.length)  
    txtFile.write(line)
```

Your final task is to close .txt File and provide a message stating the script is completed.

- n** Call the **close()** method on **txtFile**. No parameter is passed to the method.

- o Add a **print** statement to inform the user the script is completed, then debug and run your script.

Your final task is to verify the contents of the text file.

- p In the Python Interactive Window, press the Enter key to obtain the prompt.

- q Type: `os.popen(os.path.join(wksp, "MajorAttractions.txt"))`

The MajorAttractions.txt file will open in the default text viewer.

- r Review the contents and then close the viewer.
- s Close the script, but be sure to leave PythonWin open for the next step.

## Step 2: Buffer feature classes

In this step, you will write a script that buffers feature classes with the buffer distance determined by the type of geometry. For point features, the buffer distance will be set to one value; for polyline features, the buffer distance will be set to a second value; for polygon features, the buffer distance will be set to a third value.

The detailed workflow for the script is:

- Import modules and set the current workspace to the geodatabase or folder.
  - Create a list of feature classes in the workspace.
  - Using a for-in loop, process each feature class.
  - For each feature class, obtain the shapeType property from a Describe object.
  - Based on the shapeType, buffer points by 1000 ft, polygons by -750 ft, and polylines by 500 ft.
  - Confirm results in ArcMap.
- a Create a new script named **MyBufferSanDiegoFC.py**, save it to the C:/Student/PYTH/Automating\_scripts folder, then add a header to your script.
  - b Import the ArcPy module and set the workspace environment to **C:/Student/PYTH/Automating\_scripts/SanDiego.gdb**.

Your first task is to create a list of feature class names.

- c Return to the ArcGIS 10.1 Help Library topic on *Listing data* to answer the next question.

2. Which of the ArcPy functions returns a list of feature classes in the current workspace?

---

- d Using your answer, write code to use the `arcpy.List` function that creates a list of feature class names in the **SanDiego** geodatabase, assigning the list returned from the function to a variable named **fc\_list**.
- e Create a **for-in** loop on **fc\_list**, with the item value being passed to a variable named **featClass**.
- f Inside the loop, pass **featClass** as the parameter value to the **arcpy.Describe()** function. Assign the describe object returned from the function to a variable named **desc**.
- g While still in the loop, write code using the **if-elif** statement to set a variable named **buffDist** to the value:
- if **desc.shapeType** is equal to **'Point'** the **buffDist** is **'1000 feet'**
  - elif **desc.shapeType** is equal to **'Polyline'** the **buffDist** is **'500 feet'**
  - elif **desc.shapeType** is equal to **'Polygon'** the **buffDist** is **'-750 feet'**
- h Below the **if-elif** statement, and while still in the loop, add code to the script to run the **arcpy.Buffer\_analysis()** tool, setting these parameters:
- **in\_features = featClass**
  - **out\_feature\_class = featClass + "\_Buff"**
  - **buffer\_distance\_or\_field = buffDist**



There are three ways to skip optional parameters for a tool:

- Set the optional parameter to an empty set of quotations ("").
- Set the optional parameter to a pound sign in quotations ("#").
- Specify the parameter name to use along with its value.

If there is a default value for an optional parameter and the parameter is skipped, the default value will be used. A parameter's default value is detailed in the tool's help.

- i Verify your code looks like the following:

```
import arcpy
arcpy.env.workspace = "C:/Student/PYTH/Automating_scripts/SanDiego.gdb"
fc_list = arcpy.ListFeatureClasses()
```

```

for featClass in fc_list:
    desc = arcpy.Describe(featClass)

    if desc.shapeType == 'Point':
        buffDist = '1000 feet'
    elif desc.shapeType == 'Polyline':
        buffDist = '500 feet'
    elif desc.shapeType == 'Polygon':
        buffDist = '-750 feet'

    arcpy.Buffer_analysis(in_features = featClass,
                          out_feature_class = featClass + '_Buff',
                          buffer_distance_or_field = buffDist)

```

- j Outside of the loop, add a line that prints a message stating the script is completed.
- k Debug and run your script.

It will take a few moments to run.

- l If you encounter any errors:
  - Open the ArcCatalog application and navigate to the C:\Student\PYTH\Automating\_scripts folder in the tree view.
  - Open the SanDiego geodatabase and delete any feature classes created by the script.
  - Fix any errors in your script, then run the script again.
- m Close the script and PythonWin.

### Step 3: Verify script results

The final step in the process of automating your workflow is to verify that the script created valid and correct results.

- a In ArcMap, open the Catalog window and navigate to the Automating\_scripts folder.
- b Expand the SanDiego geodatabase and verify that the script created the buffered feature classes.
- c From the Catalog window, drag some of the buffered feature classes onto the map.
- d Using the tools available in ArcMap, verify that the script buffered the features with the correct distance.
- e Close ArcMap when you are done. If you would like, you can save the mxd.

## Lesson review

1. What do all ArcPy List functions return?

---

---

2. How do you iterate through a list returned by the List function?

---

---

3. Write a line of code that returns a list of integer fields from a table.

---

---

---



## Answers to Lesson 3 questions

### Explore the ArcPy List functions (page 3-5)

1. The `ListFeatureClasses()` function returns a list of items. What data type is returned from the function? What do the values contain?

**The return data type is a string. The values contain feature class names.**

2. The `ListFields()` function returns a list of items containing field objects. What are some of the field object properties that you can access?

**Field object properties returned include name, aliasName, isNullable, type, required, length, precision, scale, domain, editable, and baseName.**

3. Provide a list of ArcPy List functions that must have the current workspace environment set before the function can be used.

**ListDatasets(), ListFeatureClasses(), ListTables(), ListFiles(), ListRasters(), and ListWorkspaces().**

### Exercise 3: Automate scripts with the ArcPy List functions (page 3-9)

1. Which of these functions returns a list of fields?

**ListFields(dataset, wild\_card, field\_type) returns a list of fields.**

2. Which of the ArcPy functions returns a list of feature classes in the current workspace?

**ListFeatureClasses(wild\_card, feature\_type) returns a list of feature classes in the current workspace.**



# 4

## Working with Selections

### Key terms

feature class

feature layer

FieldInfo object

table

table view

### Introduction

Automating the process of creating new geodatabases, feature classes, and tables can be advantageous when designing, testing, and implementing a new GIS system. You can easily make changes to schema, field names, and other components as needed.

When faced with the task of needing to create a new feature class that contains a subset of data from a larger dataset, or needing to change field names in the new feature class, what options do you have?

- Use ArcCatalog and make the changes manually.
- Create a model or script to automate the process.
- Create a subset of data by making a feature layer in memory, optionally applying a SQL expression and/or a FieldInfo object to alter the schema, and then copying the feature layer to a new feature class.

### Topics covered

- Creating a feature layer or table view
- Working with a selection
- The FieldInfo object
- Applying a SQL expression to a feature layer

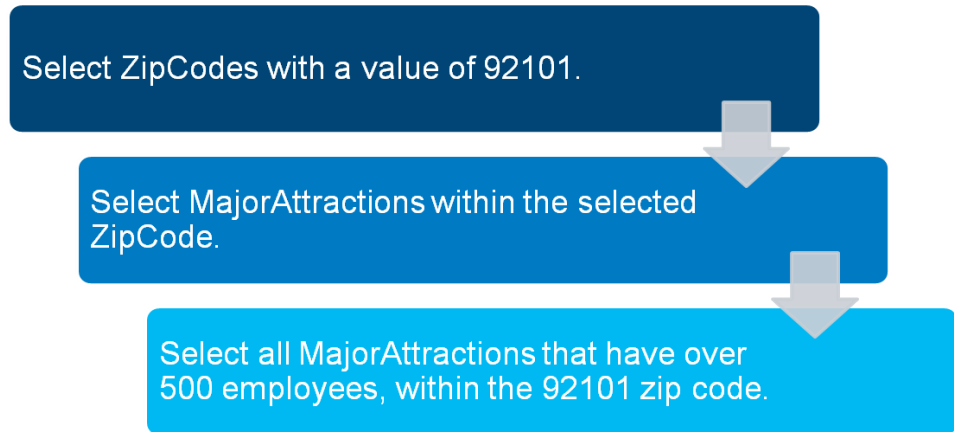
### Learning objectives

- Determine when to use a feature layer and when to use a feature class in a script.
- Choose when to create a selection from attributes and when to create a selection from spatial relationships.
- Apply a spatial selection or attribute selection to geoprocessing tools.
- Alter feature layer schema using the FieldInfo object.

## Selection tools in ArcMap

Imagine that you have been tasked to figure out how many major attractions in San Diego, California have over 500 employees. Here is a typical workflow that could be followed to answer the question using ArcMap.

**Figure 4.1**  
Workflow for typical  
selection task in ArcMap.



## Terms commonly used when working with selections

Table 4.1

| Term                    | Description                                                                                                                           |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <b>feature class</b>    | A table containing an attribute field that stores geometry that defines the shape of a feature.                                       |
| <b>feature layer</b>    | An in-memory representation of the data in a Feature Class.                                                                           |
| <b>FieldInfo object</b> | An object that provides methods and properties for working with fields. Applied to a Feature Layer or Table View to alter the schema. |
| <b>Table</b>            | A storage container for rows that contain fields to store data.                                                                       |
| <b>TableView</b>        | An in-memory representation of the data in a Table.                                                                                   |

## Tools that accept Feature Layers

Many geoprocessing tools work with or create selections on data. These tools accept only feature layers or table views as input and in *some* cases, they will also accept raster layers. For example, the SelectLayerByAttribute tool in the Data Management toolbox requires a feature layer, table view, or raster layer as input. When writing a script, you must take into account these tools that do not accept Feature Classes or tables as input.

You will use two help topics in the ArcGIS 10.1 for Desktop Help to research the geoprocessing tools that create and manipulate feature layers, layer files, and table views. You will use the information to fill out the table and answer the questions on the following page.



Use the search field to search for these topics in the help.

In the ArcGIS 10.1 Desktop Help Library, refer to the following help topics to fill out the table and answer the questions:

- *An overview of the Layers and Table Views toolset*
- *Creating and using layer selections*

1. Complete the following table. (The first row has been filled out for you.)

| Tools that create or manage feature layers and table views. | Creates output feature layer or table view? | Requires feature layer or table view for input? |
|-------------------------------------------------------------|---------------------------------------------|-------------------------------------------------|
| 1. <i>Make Feature Layer</i>                                | <i>Yes</i>                                  | <i>No</i>                                       |
| 2.                                                          |                                             |                                                 |
| 3.                                                          |                                             |                                                 |
| 4.                                                          |                                             |                                                 |
| 5.                                                          |                                             |                                                 |
| 6.                                                          |                                             |                                                 |
| 7.                                                          |                                             |                                                 |

Table 4.2

2. What is the main benefit of using a feature layer as input to a geoprocessing tool?
- \_\_\_\_\_
3. What geoprocessing tool can you use to make a feature layer permanent?
- \_\_\_\_\_

## Working with a selection

A selection can be simply defined as a subset of features from a feature class or rows from a table. When you need to work with a selection or subset of features in your script, use the `MakeFeatureLayer` tool to create the feature layer. Once the feature layer is created, you can perform additional selections as needed, or simply pass it to a geoprocessing tool.

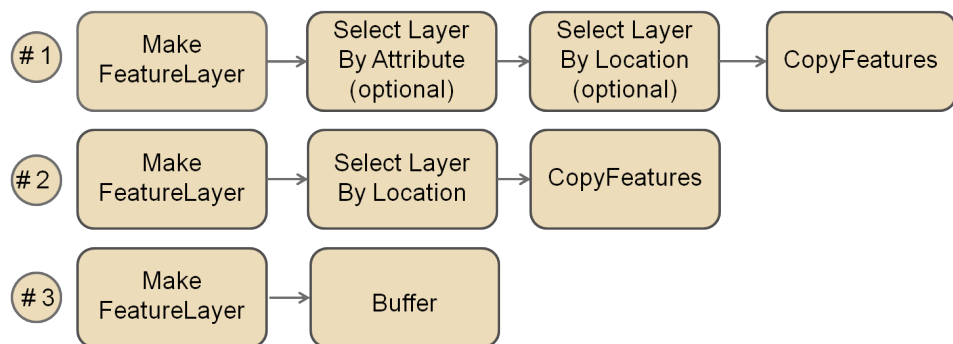
There are several workflows that can be followed when working with a selection:

**Workflow #1:** Create a feature layer using the `MakeFeatureLayer` tool, optionally perform attribute and/or spatial selections on the feature layer, and then pass it to a geoprocessing tool.

**Workflow #2:** Create a feature layer using the `MakeFeatureLayer` tool, perform a spatial selection on the feature layer, and then pass it to a geoprocessing tool.

**Workflow #3:** Create a feature layer using the `MakeFeatureLayer` tool, and then pass it directly to the geoprocessing tool.

**Figure 4.2**  
Workflows for working  
with a selection.



Because a feature layer is an in-memory representation of the data, the feature layer will persist *only* during the session. To make the feature layer permanent, you can use the `CopyFeatures_management` tool. The `CopyFeatures` tool accepts a feature layer as input and outputs a new feature class.



## The MakeFeatureLayer tool

The MakeFeatureLayer tool creates a feature layer in memory from an input feature class or layer file. The feature layer will not persist after the session ends, but can be saved to a new feature class with the CopyFeatures tool or as a layer file with the SaveToLayerFile tool.

The MakeFeatureLayer tool has optional parameters to accept a SQL expression as a where clause, a specified workspace to validate field names in the feature layer, and a FieldInfo object to alter names and/or visibility for specified fields.

**Tool syntax:**

```
arcpy.MakeFeatureLayer_management(in_features,
                                  out_layer, {where_clause},
                                  {workspace}, {field_info})
```

Table 4.3

The MakeFeatureLayer tool

| Parameter    | Description                                                                 | Data Type            |
|--------------|-----------------------------------------------------------------------------|----------------------|
| in_features  | The input feature class or layer                                            | Feature layer        |
| out_layer    | In-memory feature layer                                                     | Feature layer        |
| where_clause | SQL expression used to create a subset of features                          | SQL expression       |
| workspace    | Used to validate field names in the in-memory feature layer                 | Workspace or dataset |
| field_info   | Used to alter field names and/or hide fields in the in-memory feature layer | FieldInfo            |



Sample scripts for working with the `MakeFeatureLayer` tool can be found in the `C:\Student\PYTHON\Selections\Samples` folder.



ArcGIS 10.1 Help Library:  
*Make Feature Layer (Data Management)*

## Working with the `MakeFeatureLayer` tool

- Only simple feature classes are supported. Annotation and dimension feature classes are *not* supported.
- A SQL expression with invalid syntax will result in an empty feature layer.
- If a folder path is specified for the workspace parameter, field names will be truncated to 10 characters.
- Use the `Delete_management` tool if you need to recreate the feature layer or run your script an additional time. Attempting to run the `MakeFeatureLayer` tool and specifying the `out_layer` parameter value to be the same as an existing feature layer will create a runtime exception.



ArcGIS 10.1 Help Library:  
*FieldInfo (arcpy)*



Sample scripts for working  
with the `FieldInfo` object  
and the  
`MakeFeatureLayer` tool  
can be found in the `..\PYTH\Selections\Samples`  
folder.

Table 4.4

## The FieldInfo object

The `FieldInfo` object can be used in conjunction with the `MakeFeatureLayer` tool to alter field names and visibility for the newly created feature layer. In addition to these capabilities, a split rule can also be applied to any field in the feature layer.

The `FieldInfo` object provides methods and properties for layers and table views, and can be passed as an optional parameter to the `MakeFeatureLayer` tool or the `MakeTableView` tool. The `Describe` object returned from the `arcpy.Describe` function will provide the `FieldInfo` object for a feature layer or table view if defined when the layer or view were created.

### Syntax:

```
arcpy.FieldInfo()
```

**arcpy.FieldInfo properties**

| Property | Description                        | Data Type |
|----------|------------------------------------|-----------|
| count    | The number of fields to be altered | Integer   |

Method:

```
addField(field_name, new_field_name, visible,
        split_rule)
```

Table 4.5

FieldInfo object parameters for the addField method

| Parameter      | Description                                                                                                                                                                                |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| field_name     | Field name to alter                                                                                                                                                                        |
| new_field_name | New field name to apply                                                                                                                                                                    |
| visible        | <ul style="list-style-type: none"><li>• "HIDDEN" to remove field from output feature layer or table view</li><li>• "VISIBLE" to keep field in output feature layer or table view</li></ul> |
| split_rule*    | <ul style="list-style-type: none"><li>• "NONE" retains field values</li><li>• "RATIO" computes new field values based on division of feature geometry</li></ul>                            |

\*The *split rule* is honored by geoprocessing tools in your script that use the feature layer as input. When the split rule is set to "RATIO", if the geometry of features are split during processing, the attributes of the resulting features are calculated based on the ratio of the input geometry to the new feature geometry.

For example, if an input feature was split in a 70/30 ratio, each new feature's attribute values will be either 70% of the original feature value or 30% of the original feature value. The ratio option will work only on numeric fields.



ArcGIS 10.1 Help Library:  
*AddFieldDelimiters (arcpy)*

## The AddFieldDelimiters function

The `arcpy.AddFieldDelimiters` function can be thought of as a helper-type function. Its task is to return a field name with the proper field delimiters for the workspace specified.

### Tool syntax:

```
arcpy.AddFieldDelimiters(datasource, field)
```

Table 4.6

**arcpy.AddFieldDelimiters function**

| Parameter  | Description                                                                  |
|------------|------------------------------------------------------------------------------|
| datasource | The input workspace, dataset, feature class, layer, table, or SDE connection |
| field      | The field name for which the field delimiters will be added.                 |

Table 4.7

**Example uses of arcpy.AddFieldDelimiters**

| Datasource                                                                                                                   | Delimiter       | Example          |
|------------------------------------------------------------------------------------------------------------------------------|-----------------|------------------|
| <ul style="list-style-type: none"> <li>• Shapefile</li> <li>• File geodatabase</li> <li>• CAD</li> <li>• Coverage</li> </ul> | " "             | "NAME" = 'China' |
| <ul style="list-style-type: none"> <li>• Personal geodatabase</li> </ul>                                                     | [ ]             | [NAME] = 'China' |
| <ul style="list-style-type: none"> <li>• ArcSDE geodatabase</li> </ul>                                                       | no<br>delimiter | NAME = 'China'   |

## Determining a workflow

When writing a script that uses feature layers and selections, it may be beneficial to break the script down into sections in order to determine the workflow.

### Sample scenario:

The City of Wilson has been asked to provide a count of all alcohol-related crimes near the Historical District. The parameters provided to perform the tasks are:

- Locate all of the restaurants that are within 1000 feet of the Historical District that serve alcohol.
- Locate all alcohol related crimes that are within 500 feet of the selected restaurants.
- Report on the number of selected alcohol-related crimes.

The script workflow can be broken into five major areas:

**Figure 4.3**

Script workflow for locating crimes within 500 feet of restaurants that serve alcohol and are within 1000 feet of the City's Historical District.



### Steps required for the script:

- Import the ArcPy site package and set the current workspace.
- Use `arcpy.MakeFeatureLayer` to create three feature layers: Crimes96, Restaurants, and Historical Districts.
- Use `arcpy.SelectLayerByLocation` to select all restaurants within 1000 feet of the Historical Districts.
- Use `arcpy.SelectLayerByLocation` to select all crimes within 500 feet of the selected restaurants.
- Use `arcpy.SelectLayerByAttribute` to create a subset of selected crimes where the ALCOHOL field has a value greater than 0.
- Use `arcpy.GetCount` to obtain a count of selected crimes.
- Print the selected count of alcohol-related crimes.

## Creating feature layer and get feature count

### Tips for working with feature layers

- When working with geoprocessing tools in your scripts, you can skip optional parameters by specifying each parameter name and the value to assign. Some tools may need to have a mix of required and optional parameters filled in, but not all optional parameters may be needed for the desired tool result.
- The MakeFeatureLayer and MakeTableView tools are aware of and work with layer selections when the tool is run in ArcMap.
- Feature layers and table views can be used as inputs to many geoprocessing tools.
- Feature layers and table views are temporary by nature and will not persist after the session ends.
- Use the `arcpy.Delete_management` tool to remove a feature layer or table view from memory. If the feature layer or table view has been added to ArcMap's Table Of Contents window, the tool will also remove the layer or view from ArcMap.





50 minutes



## Exercise 4: Work with Feature Layers and Selections

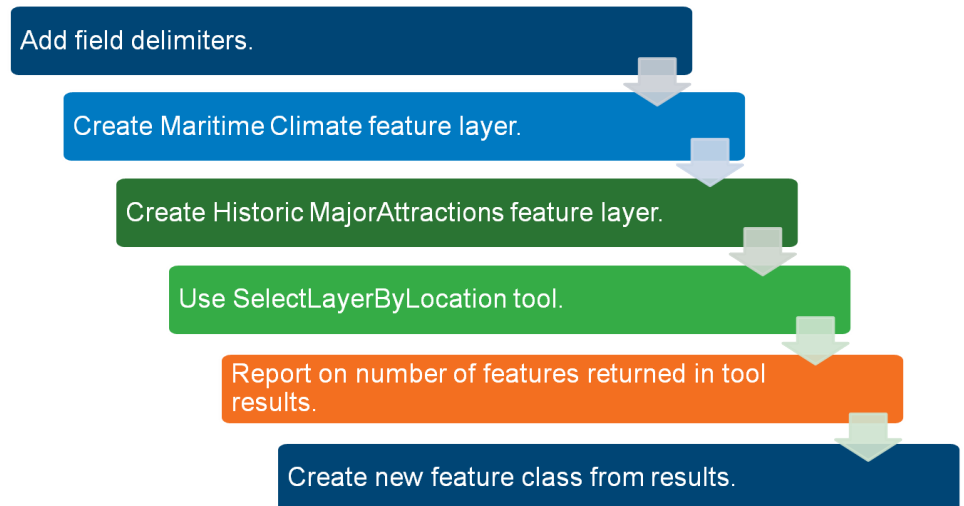
The City of San Diego has received a request to find all historic MajorAttractions that are within the maritime climate zones. These historic MajorAttractions will be placed in a separate feature class to be used in a project where a special map will be created for reporting purposes.

Your tasks are to determine the historic attractions, overlay them with the maritime climate zones, and then copy the results to a new feature class.

In this exercise, you will:

- Create feature layers, applying SQL expressions and altering field names.
- Apply a spatial selection between the feature layers.
- Copy the results to a new feature class.

**Figure 4.4**  
High-level workflow for  
exercise.



## Step 1: Create new script

In this step, you will create a new script in PythonWin, add a header to the script, and add code to prepare for creating the Climate and MajorAttractions feature layers.

Workflow for step:

- Create new script.
- Import modules and set current workspace.
- Add field delimiters to fields used in SQL expressions.
- Assemble SQL expressions for use in creating feature layers.

- In PythonWin, create a new Python script named **MyCreateMaritimeAttractions.py** and save it in the C:\Student\PYTH\Selections folder.
- Add a header to the script, then document the script name and purpose.
- Import the ArcPy module and set the current workspace to **C:/Student/PYTH/Selections/SanDiego.gdb**.

Your next task in the workflow is to add field delimiters to the TYPE and ESTAB fields. These fields will be used in constructing two SQL expressions.



When working with geoprocessing tools in your scripts, you can skip optional parameters by specifying each parameter name and the value to assign. Some tools will have a mix of required and optional parameters specified, but not all optional parameters may be needed for the desired tool result.

- Add code to the script using the `arcpy.AddFieldDelimiters` function with these parameters:
  - **arcpy.env.workspace** for the dataset parameter
  - **"TYPE"** for the field parameter.
- Assign the return from the tool to a variable named **newField1**.
- Add code to use the **arcpy.AddFieldDelimiters** function a second time with these parameters:
  - **arcpy.env.workspace** for the first parameter
  - **"ESTAB"** for the second parameter.

- g** Assign the return from the tool to a variable named **newField2**.

The next task is to assemble the two SQL expressions, which will be used later in the script as the **where\_clause** parameters for the **arcpy.MakeFeatureLayer\_management** function.

The first SQL expression will be used for creating the Maritime feature layer, and the second SQL expression will be used to create the Historic Attractions feature layer.

- h** Add these two lines of code to the script:

```
maritimeSQLExp = newField1 + " = " + " 'Maritime' "
historicSQLExp = newField2 + " > 0 and " + \
    newFldName2 + " < 1956"
```

In the next exercise step, you will apply the SQL expressions to the **arcpy.MakeFeatureLayer\_management** function.

- i** Save your script and check for syntax errors using the Check button.

## Step 2: Create Feature Layers

In this exercise step, you will add code to your script to create the Maritime Climate feature layer and the Historic Attractions feature layer.

Workflow for step:

- Create feature layer from Climate, applying the first SQL expression.
- Create feature layer from MajorAttractions, applying the second SQL expression.

- a** In your script, add code to create the Maritime layer using the **arcpy.MakeFeatureLayer\_management** function, passing in these parameters:

- **"Climate"** for the **in\_features** parameter
- **"MaritimeLyr"** for the **out\_layer** parameter
- **maritimeSQLExp** variable for the **where\_clause** parameter

- b** Add code to create the Historic Attractions layer using the **arcpy.MakeFeatureLayer\_management** function, passing in these parameters:

- **"MajorAttractions"** for the **in\_features** parameter
- **"HistoricLyr"** for the **out\_layer** parameter
- **historicSQLExp** variable for the **where\_clause** parameter

- c Check your script for syntax errors and fix any that are detected.

### Step 3: Perform Spatial Selection

In this exercise step, you will perform a spatial selection using the `arcpy.SelectLayerByLocation` function to find all Historic Attractions that are contained by the Maritime Climate features. This will provide you with a result that can be saved to a new feature class in the final exercise step.

Workflow for step:

- Use the `arcpy.SelectLayerByLocation` function to perform a spatial selection. The input features will be the historic attractions, the selecting features will be the maritime climate, the overlap type will be `COMPLETELY_WITHIN`, and the tool will return a `NEW_SELECTION`.
  - Confirm that the correct number of features have been selected.
- a Add code to call the `arcpy.SelectLayerByLocation_management` function, passing in the parameters:
    - `"HistoricLyr"` for the `in_layer` parameter
    - `"COMPLETELY_WITHIN"` for the `overlay_type` parameter
    - `"MaritimeLyr"` for the `select_features` parameter
    - `" "` for the `search_distance` parameter
    - `"NEW_SELECTION"` for the `selection_type` parameter
  - b Add code to call the `arcpy.GetCount_management` function on `"HistoricLyr"`, assigning the return to a variable named `featCount`.
  - c Add code to print the number of selected features, which is the value in `featCount`.
  - d Check your script for syntax errors.



If you would like to check for valid results, you can run your script at this time. Running the script should result in a selected feature count of eight.

Remember that the feature layers will remain in memory as long as the PythonWin session is running, or until you use the `arcpy.Delete_management` function to remove them.

## Step 4: Create Feature Class from selection

For the final step in the exercise, you will write code to store the selected features in the Historic Attractions feature layer to a new feature class named MaritimeAttractions.

For the final task in the script, the feature layers will be removed from memory.

Workflow for step:

- Use the `CopyFeatures_management` function to store selected features in new feature class.
  - Use the `Delete_management` function to remove the feature layers from memory.
- a In your script, add code to call the **`arcpy.CopyFeatures_management`** function, passing in these parameters:
    - **`"HistoricLyr"`** for the `in_features` parameter
    - **`"MaritimeAttractions"`** for the `out_feature_class` parameter
  - b Add code to call the **`arcpy.Delete_management`** function twice, once for each feature layer created in the script. Pass the name of the feature layer to the tool's parameter.
  - c Add a final line of code to print that the script has completed.
  - d If you ran your script at the end of Step 3, type these two commands at the Interactive Window to remove the feature layers in memory prompt:  

```
arcpy.Delete_management("MaritimeLyr")  
arcpy.Delete_management("HistoricLyr")
```
  - e Debug and run your script, and then use ArcCatalog to verify that the MaritimeAttractions feature class has been created.
  - f Close PythonWin and ArcCatalog.

## Lesson review

1. List two geoprocessing tools that must use a feature layer to create selections.

---

---

2. What kind of schema changes can you make on a field with a FieldInfo object?

---

---

---

3. If a field is set to HIDDEN in a FieldInfo object, will the field be available in the output feature layer?

---

---

---

## Answers to Lesson 4 questions

### Tools that accept Feature Layers (page 4-4)

1. Complete the following table. (The first row has been filled out for you.)

| Tools that create or manage feature layers and table views. | Creates output feature layer or table view? | Requires feature layer or table view for input? |
|-------------------------------------------------------------|---------------------------------------------|-------------------------------------------------|
| 1. <i>Make Feature Layer</i>                                | <i>Yes</i>                                  | <i>No</i>                                       |
| 2. <b>Make Query Table</b>                                  | <b>Yes</b>                                  | <b>No</b>                                       |
| 3. <b>Make Table View</b>                                   | <b>Yes</b>                                  | <b>No</b>                                       |
| 4. <b>Make XY Event Layer</b>                               | <b>Yes</b>                                  | <b>No</b>                                       |
| 5. <b>Save To Layer File</b>                                | <b>No</b>                                   | <b>Yes</b>                                      |
| 6. <b>Select Layer By Attribute</b>                         | <b>No</b>                                   | <b>Yes</b>                                      |
| 7. <b>Select Layer By Location</b>                          | <b>No</b>                                   | <b>Yes</b>                                      |

2. What is the main benefit of using a feature layer as input to a geoprocessing tool?

**Only the selected features will be used.**

3. What geoprocessing tool can you use to make a feature layer permanent?

**The CopyFeatures\_management tool.**





# 5

## Working with Cursors

### Key terms

cursor objects

insert cursor

search cursor

update cursor

### Introduction

Accessing data in feature classes and tables is the starting point for many data management and analysis workflows. A cursor is a data access object that can be used to iterate over a set of data in a table or insert new rows into a table. Many workflows include adding new data, updating existing data, or working with a subset of data to limit which feature values are accessed and updated. The cursor allows you to script and automate these types of data access tasks.

In the previous lesson, you learned about working with selections and subsets of data, which are used in geoprocessing tools and to create new feature classes. In this lesson, you will work with the da cursors to read, update, and populate feature classes and tables.

### Topics covered

- arcpy.da cursors
- Search cursor
- Update cursor
- Insert cursor
- Classic ArcPy cursors

### Learning objectives

After completing this lesson, you will be able to:

- Work with arcpy.da cursors to read, update, and populate feature classes.
- Incorporate best practices into your workflows for working with cursors.

## The arcpy.da cursors

*What are some reasons you would want to write a script to update field values?*

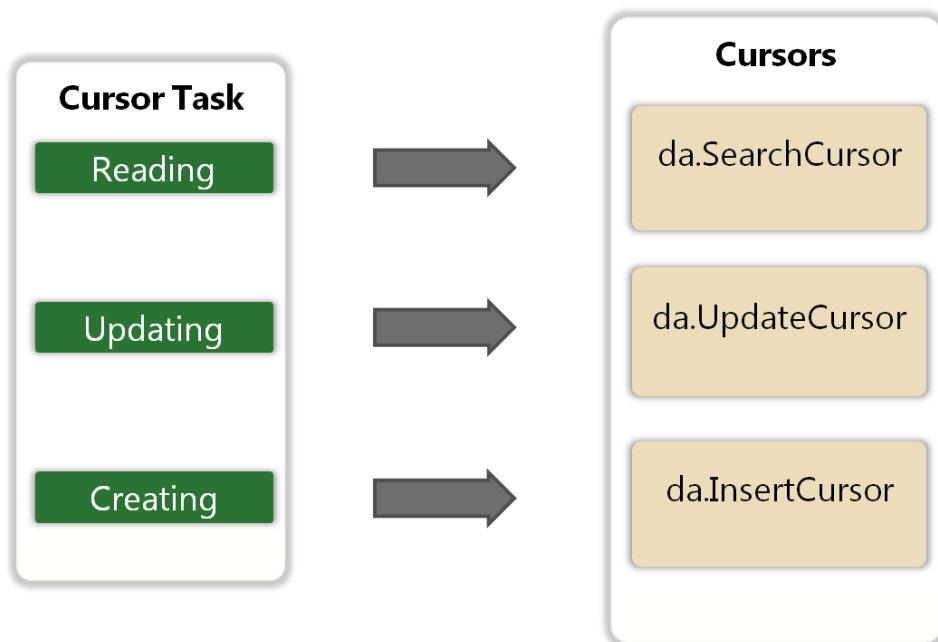


ArcGIS 10.1 Help Library:  
*What is the data access  
module?*

**Figure 5.1**

The arcpy.da cursor tasks.

Throughout this lesson, you will explore working with ArcPy **cursor objects**, which are accessed from the arcpy.da module. These cursors are designed to allow high-performance access to field contents in feature classes and tables, update field contents, and populate new rows.



There are several cursor methods available, depending on the cursor being used.

Table 5.1

### Cursor object methods

| Cursor methods | Description                                                                                                                                            | Supported by                 |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|
| next()         | Returns the next row from the cursor as a tuple. The order of the values is returned in the order of the fields specified when the cursor was created. | SearchCursor<br>UpdateCursor |
| reset()        | Resets the cursor back to the first row to allow for multiple passes over the data.                                                                    | SearchCursor<br>UpdateCursor |
| updateRow()    | Updates the current row.                                                                                                                               | UpdateCursor                 |
| deleteRow()    | Deletes the current row. Use this with caution.                                                                                                        | UpdateCursor                 |
| insertRow()    | Inserts new rows in the feature class or table. A tuple of values is passed to the method in the order of the fields specified for the cursor.         | InsertCursor                 |



Using a `with` statement guarantees that database locks are released and the cursor iteration is reset.

## Using the SearchCursor

The `arcpy.da.SearchCursor` provides read-only access to field values in a feature class or table and a feature layer or table view. It can be iterated by using a `for-in` statement and a `with` statement.

The `SearchCursor` records can be filtered to match an attribute field where clause and a SQL clause when working with a geodatabase data source, such as a feature class or table.

### SearchCursor syntax:

```
arcpy.da.SearchCursor(in_table, field_names,
                      (where_clause), {spatial_reference},
                      (explode_to_points), {sql_clause})
```

Table 5.2

**arcpy.da SearchCursor parameters**

| Parameter         | Description                                                      |
|-------------------|------------------------------------------------------------------|
| in_table          | A feature class, table, feature layer, or table view             |
| field_names       | A tuple or list of field names; SHAPE@ tokens are also supported |
| where_clause      | Attribute field SQL expression                                   |
| spatial_reference | The Spatial Reference of the feature class                       |
| explode_to_points | Deconstructs a feature into its vertices or points               |
| sql_clause        | SQL prefix and postfix clauses                                   |

Table 5.3

**SearchCursor methods**

| method | Description                                                                                                         |
|--------|---------------------------------------------------------------------------------------------------------------------|
| next() | Returns the next row in the cursor as a tuple of field values in the order the fields were specified in the cursor. |
| reset  | Resets the cursor to the first row.                                                                                 |

Accessing the SHAPE field can impact the performance of the arcpy.da cursors. Tokens can be specified for the Shape field without imposing a major performance penalty. Tokens will access Shape geometry properties without having to actually work with the entire Shape geometry for a feature. An exception is the SHAPE@ token. It will return the full geometry for the feature and can impact cursor performance. It can be used for accessing geometry properties, such as the extent of the feature, the type of feature, and its point or part count.

Table 5.4

**Shape field tokens**

| Token              | Description                                    |
|--------------------|------------------------------------------------|
| OID@               | The ObjectID field value                       |
| SHAPE@X            | The feature's X coordinate                     |
| SHAPE@Y            | The feature's Y coordinate                     |
| SHAPE@XY           | The feature's X&Y coordinates (tuple)          |
| SHAPE@TRUECENTROID | The feature's centroid X&Y coordinates (tuple) |
| SHAPE@M            | The feature's M value                          |
| SHAPE@Z            | The feature's Z value                          |
| SHAPE@AREA*        | The feature's area                             |
| SHAPE@LENGTH*      | The feature's length                           |
| SHAPE@             | The geometry of the feature                    |

\*Should be considered read-only properties and cannot be used with an Update or Insert cursor.



ArcGIS 10.1 Help Library:  
*SearchCursor (arcpy.da)*



Open the *Python modules documentation*, click on *Python v2.7 documentation*, scroll down on the page to *5.1.4 List Comprehensions* and click on the link.

## Notes and Tips for the `da.SearchCursor`

- The Search Cursor provides read-only access to field values.
- The Shape field properties can be efficiently accessed with the `SHAPE@` tokens. Returning the full geometry from the Shape field is an expensive operation for the cursor and should be avoided if possible.
- Cursors can only be iterated in a forward direction. There is no support for backing up or for retrieving rows that have already been retrieved. You can use the `reset()` method to return to the first row in the cursor if multiple passes over the data is necessary.
- The list of tuple field values that is returned from the `arcpy.da` cursors can be sorted using the Python `sorted()` method. The `sorted` method returns an ascending sort by default. For a descending sort, set the `reverse` parameter to `True`.
- It can be advantageous to return a sorted set of unique values from a field. You can use a list comprehension on the `SearchCursor` to obtain a Python List of field values, use the `set()` function to obtain the unique values, and then use the `sorted()` function to return the unique values sorted in ascending or descending values.



Using a `with` statement guarantees that any database locks are released and the cursor iteration is reset.

## Using the UpdateCursor

The **`arcpy.da.UpdateCursor`** provides read-write access to field values in a feature class or table, as well as a feature layer or table view. UpdateCursors can be iterated by using a `for-in` statement or a `with` statement.

The UpdateCursor records can be filtered to match an attribute field where clause, and a SQL clause when working with a geodatabase data source, such as a feature class or table.

### UpdateCursor syntax:

```
arcpy.da.UpdateCursor(in_table, field_names,
                      (where_clause), {spatial_reference},
                      (explode_to_points), {sql_clause})
```

### arcpy.da.UpdateCursor parameters

Table 5.5

| Parameter                      | Description                                                      |
|--------------------------------|------------------------------------------------------------------|
| <code>in_table</code>          | A feature class, table, feature layer, or table view             |
| <code>field_names</code>       | A tuple or list of field names; SHAPE@ tokens are also supported |
| <code>where_clause</code>      | Attribute field SQL expression                                   |
| <code>spatial_reference</code> | The Spatial Reference of the feature class                       |
| <code>explode_to_points</code> | Deconstructs a feature into its vertices or points               |
| <code>sql_clause</code>        | SQL prefix and postfix clauses                                   |

Table 5.6

UpdateCursor methods

| method         | Description                        |
|----------------|------------------------------------|
| updateRow(row) | Updates the current row            |
| deleteRow()    | Deletes the current row            |
| next()         | Returns the next row in the cursor |
| reset()        | Resets the cursor to the first row |

Tips for the da.UpdateCursor

- When updating the current row in the cursor, be sure to call the `updateRow(row)` method before fetching the next row from the cursor. The row object passed to the `updateRow()` method is actually a tuple of values. The order of the values must be in the same order as the fields specified in the cursor.
- Use the `SHAPE@XY` token to efficiently update feature geometry for a point feature class.



ArcGIS 10.1 Help Library:  
*UpdateCursor (arcpy.da)*



## Using the InsertCursor

The **arcpy.da.InsertCursor** creates a write-only cursor to field values in a feature class or table, as well as a feature layer or table view.

### InsertCursor syntax:

```
arcpy.da.InsertCursor(in_table, field_names)
```

Table 5.7

**arcpy.da.InsertCursor parameters**

| Parameter   | Description                                                      |
|-------------|------------------------------------------------------------------|
| in_table    | A feature class, table, feature layer, or table view             |
| field_names | A tuple or list of field names; SHAPE@ tokens are also supported |

Table 5.8

**InsertCursor method**

| method         | Description                      |
|----------------|----------------------------------|
| insertRow(row) | Inserts a new row into the table |



ArcGIS 10.1 Help Library:  
*InsertCursor (arcpy.da)*

### Cursor table locking

The InsertCursor will honor existing table locks that are set by ArcGIS applications. Locks are used in ArcGIS to prevent multiple processes from changing the same table at the same time.

Two types of locks are possible: *shared* and *exclusive*.

#### Shared lock

- Placed on a table any time a table or feature class is accessed by ArcGIS.
- Examples of shared locks: Displaying a feature class in ArcMap or previewing a table in ArcCatalog.



An Insert cursor cannot be created on a feature class or table if an exclusive lock is already applied.

## Exclusive lock

- Applied when changes are made to a table or feature class, either to field values or to the schema.
- Examples of when an exclusive lock is applied: Using an UpdateCursor or InsertCursor in a Python IDE, saving edits to a feature class in ArcMap, or changing the schema in ArcCatalog.
- Update cursors and Insert cursors apply an exclusive lock if successfully created.
- Always del the InsertCursor and UpdateCursor objects to remove the exclusive database lock.
- An ArcMap edit session places a shared lock on the table during the edit session. When edits are saved, an exclusive lock is placed on the table through the duration of the save operation. If a Python script has an exclusive lock on the data, an edit session cannot be started in ArcMap.

## Notes and Tips for the da.InsertCursor

- When inserting new features into a Point feature class, it is more efficient to use the `SHAPE@XY` token than to create a PointGeometry object. The PointGeometry object would need to be set to the `SHAPE@` token, which is a more expensive insert operation.
- If the feature class supports M & Z values, use the `SHAPE@M` and `SHAPE@Z` tokens to specify these values for the feature.
- Any field that is not included in the InsertCursor field names list will be automatically assigned the field's default value.
- Be sure the order of values in the tuple passed to the `insertRow()` method is the same as the order of fields for the InsertCursor.

## Tips and best practices for arcpy.da Cursors

- When accessing the Shape field, use the `SHAPE@` tokens for best cursor performance.
- Use the `with` statement on Search and Update cursors to guarantee closing of the cursor and release of database locks.
- Use the `sorted()` built-in function to return a sorted list of values from a cursor or a list.
- The SearchCursor, UpdateCursor, and InsertCursor objects have a `fields` property that can be used to confirm the order of field values in the tuple that is returned by the row object.
- Cursors are navigated in a forward direction only.
- Rows may be retrieved only once.
- Use the `reset()` method to return cursor to the first row and to reset the iteration for multiple passes over the data.
- Use the `del` statement to close the cursor and release database locks.
- Enclose an UpdateCursor or InsertCursor in a `try-except` statement to handle any exception that might be generated when creating the cursor. All Update and Insert cursors must be able to create an exclusive lock on the table.



75 minutes



## Exercise 5: Work with cursors

Many analysis workflows include adding a field and calculating field values. To support this workflow, you will create scripts to report existing field values, add and populate a new field, and then create a new feature class and populate an attribute field.

In this exercise, you will:

- Report field values.
- Add a new field.
- Populate the new field.
- Create a new feature class.
- Populate the feature class.

## Step 1: Research the da Cursors

In this step, you will research the SearchCursor, UpdateCursor and InsertCursor functions in the Data Access module.

- a** In the ArcGIS 10.1 Help Library, navigate to the *Accessing data using cursors* help topic.

*Hint:* Geoprocessing > Python > Accessing geographic data in Python > Working with cursors in Python > Accessing data using cursors.

- b** Click the link for each cursor in the Cursor table. Take a few moments to examine the summary, discussion, and syntax for the InsertCursor, SearchCursor, and UpdateCursor.

1. What is the function of the SearchCursor?

---

2. What does the SearchCursor return?

---

---

3. What method on the UpdateCursor can be used to remove a row from the feature class or table?

---

4. For the InsertCursor, what is passed to the row parameter of the `insertRow()` method?

---

## Step 2: Work with the da.SearchCursor

You have been tasked with printing a three-line, address-style listing of data in the MajorAttractions feature class. You will write a script to accomplish the task using the da.SearchCursor.

Workflow for this step:

- Create a new script, import required modules, and set environment values.
  - Create the SearchCursor object, passing in the feature class and field names.
  - Iterate through the rows returned by the cursor.
  - Print field values to the Interactive Window.
- a Create a new Python script named **MyAttractions.py** and save it in the C:\Student\PYTH\Cursors folder.
  - b Add a header to the script, and document the script's purpose.
  - c Add code to your script using this workflow:
    - Import the ArcPy module
    - Set the current workspace to **C:\Student\PYTH\Cursors\SanDiego.gdb**.
    - Use the **arcpy.da.SearchCursor** in a **with** statement, passing these parameters to the SearchCursor:
      - **"MajorAttractions"** for the **in\_table** parameter
      - **["NAME", "ADDR", "CITYNM", "ZIP"]** for the **field\_names** parameter
      - **cursor** for the name of the SearchCursor object, as in the form of
    - **with arcpy.da.SearchCursor() as cursor:**
    - Inside the **with** statement, create a **for-in** loop on **cursor**, returning each item to a variable named **row**
    - Print the **name**, **address**, **city**, and **zip code** for each feature returned from the **row** list. Use a three line address style:
 

```
FOUR POINTS HOTEL
8110 AERO DR
SAN DIEGO 92123
```



If you need additional assistance with the script, the MyAttractions.py exercise solution script is located in the C:\Student\PYTH\Cursors\Solution folder.

*Hint:* The `\n` can be used in the print statement as a newline character.

The list of values in the row object are in the same order as the field names passed to the cursor.

- d Check your script for syntax errors, then run the script.
- e In the Interactive Window, scroll to the end of the list of addresses.

5. What is the address for the Hilton Crystal Bay Hotel?

---

---

**f** Close the MyAttractions.py script.

### Step 3: Work with the da.UpdateCursor

Workflow for step:

- Create a new Python script, import required modules, and set the environment values.
- Determine and use the appropriate Geoprocessing tool to add a new ACRES field to the Corvallis Parcel feature class.
- Create the da.UpdateCursor object, passing in the feature class and the field names.
- Iterate through the rows in the cursor object.
- Calculate the acreage value using the `SHAPE@` token on the Shape field.
- Populate the ACRES field with the acreage value.

You have just been handed a request to add a new ACRES field to the Parcel feature class and update the values in the new field. You will write a script to perform the workflow.

**a** Using the ArcGIS 10.1 for Desktop Help, search for a geoprocessing tool that will add a new field to a feature class or table. Use your research results to help answer the following question.

6. What geoprocessing tool located in the Data Management toolbox was returned by the Desktop Help Search?

---

**b** Create a new Python script named **MyParcel\_Acres.py** and save it to C:\Student\PYTHON\Cursors folder.

**c** Add a header to the script, and document the script's purpose.



- d Write the following script:
- Import the `arcpy` module.
  - Set the workspace to the `C:\Student\PYTH\Cursors\Corvallis.gdb` file geodatabase.
  - Use the answer to the above question to add a new **ACRES** field to the **Parcel** feature class. Pass the parameters to the tool in this order:
    - **"Parcel"** for the `in_table` parameter
    - **"ACRES"** for the `field_name` parameter
    - **"DOUBLE"** for the `field_type` parameter
  - Create a `arcpy.da.UpdateCursor` on the **Parcel** feature class specifying the **ACRES** field and the **SHAPE@** token for the SHAPE field.
  - Iterate through the features in the `UpdateCursor`.
    - For each feature, obtain the Area value from the SHAPE field using the **SHAPE@** token and calculate the acres value. One acre equals 43560 square feet or 4046.856 square meters.
    - Populate the **ACRES** field with the calculated value and update the row.
  - If you did not create the cursor using a `with` statement, **del** the cursor.

If you need additional assistance with the script, the `MyParcel_Acres.py` exercise solution script is located in the `C:\Student\PYTH\Cursors\Solution` folder.

- e Check your script for syntax errors, then run the script.
- f Close the script.
- g Open ArcCatalog and view the updated table in the Corvallis geodatabase. Verify that the field has been added and the **ACRES** field is populated.

## Step 4: Work with the `da.InsertCursor`

Workflow for step:

- Create a new Python script, import required modules, and set the environment values.
- Create a Python list of values that will be used to construct new rows.
- Determine and use the appropriate geoprocessing tool to create a new feature class.
- Add the **NAME** field to the feature class.
- Create the `InsertCursor` object, passing in the fields to be populated.
- Use a list loop to populate rows in the new feature class.

Your current task is to create a new feature class containing a **"NAME"** field. **X&Y** values will be used to populate the SHAPE field. You will write the script using the `da.InsertCursor` to accomplish the task.

- a Create a new Python script named **MyPopulate\_XY.py** and save it to the C:\Student\PYTH\Cursors folder.
- b Add a header to the script, and document the script's purpose.
- c Import the `arcpy` module and set the workspace environment setting to **C:\Student\PYTH\Cursors\Corvallis.gdb**.
- d Create a Python list containing items in the form of:
 

```
[ [<Name1>, (<X1>, <Y1>)], [<Name2>, (<X2>, <Y2>)],  
  [<Name3>, (<X3>, <Y3>)], [<Name4>, (<X4>, <Y4>)] ]
```
- e Assign it to a variable named **rowValues**. Use the following table for the values.

| Name      | X       | Y     |
|-----------|---------|-------|
| Benton    | -123.40 | 44.49 |
| Linn      | -122.49 | 44.48 |
| Polk      | -123.38 | 44.89 |
| Tillamook | -123.65 | 45.45 |

- f Use the appropriate geoprocessing tool to create a new point feature class. Pass these parameters to the geoprocessing tool:
  - **arcpy.env.workspace** for the `out_path` parameter
  - **"CountyPNT"** for the `out_name` parameter
  - **"POINT"** for the `geometry_type` parameter

*Hint:* In the ArcGIS Desktop Help, search for **create feature class** to find a geoprocessing tool that can create a feature class.
- g For the new **CountyPNT** feature class, use the appropriate geoprocessing tool to add a new field, with a `field_name` of **"NAME"** and a `field_type` of **"TEXT"**.
- h Create an **arcpy.da.InsertCursor**, passing these parameters:
  - **"CountyPNT"** for the `in_table` parameter
  - **("NAME", "SHAPE@XY")** for the `field_names` parameter

- i Assign the cursor object returned from the `InsertCursor` to a variable named `iCur`.
- j Iterate through `rowValues` using a `for-in` loop, returning each item in `rowValues` to a variable named `row`.
  - Call the `insertRow()` method on `iCur`, passing the `row` object to the method's argument.
- k Be sure to `del` the cursor.

If you need additional assistance with the script, the `MyPopulate_XY.py` exercise solution script is located in the `C:\Student\PYTH\Cursors\Solution` folder.

- l Debug and run your script.
- m Verify the new feature class has been created in the Corvallis geodatabase and that it contains four rows.

## Lesson review

1. When working with the da cursors, which Python statement can be used to automatically close the cursor?
  - a. A for-in loop
  - b. A range loop
  - c. A with loop

2. Which token can be used with the da cursor to efficiently return the X&Y values of a point feature?

---

---

3. Write a single line of code to return a sorted cursor of descending LAND\_VALUE values from the Corvallis Parcel feature class.

---

---

---

## Answers to Lesson 5 questions

### Exercise 5: Work with cursors (page 5-13)

1. What is the function of the SearchCursor?

**The SearchCursor provides read-only access to the records in a feature class or table.**

2. What does the SearchCursor return?

**The SearchCursor returns an iterator of tuples. The order of the values in the tuple is the same as the order of the fields specified in the cursor's field\_names parameter.**

3. What method on the UpdateCursor can be used to remove a row from the feature class or table?

**The deleteRow() method.**

4. For the InsertCursor, what is passed to the row parameter of the insertRow() method?

**A tuple of values in the order of the fields specified in the cursor's field\_names parameter.**

5. What is the address for the Hilton Crystal Bay Hotel?

**900 F ST  
CHULA VISTA, CA 91910**

6. What geoprocessing tool located in the Data Management toolbox was returned by the Desktop Help Search?

**The Add\_Field\_management tool.**



# 6

## Working with Geometry objects

### Key terms

geometry list

geometry object

### Introduction

In many geoprocessing workflows, you will run tools that work with coordinate and geometry information. The geoprocessing tools that you will use may accept features from a feature layer or feature class as input to the tool. These same tools will also accept geometry objects as input in place of features.

Geometry objects that can be created in memory include Geometry, MultiPoint, PointGeometry, Polyline, and Polygon. These can be empty geometry objects or can be populated with coordinate pairs of X,Y values.

### Topics covered

- Creating geometry objects
- Accessing and updating geometry
- Using geometry objects in a geoprocessing tool

### Learning objectives

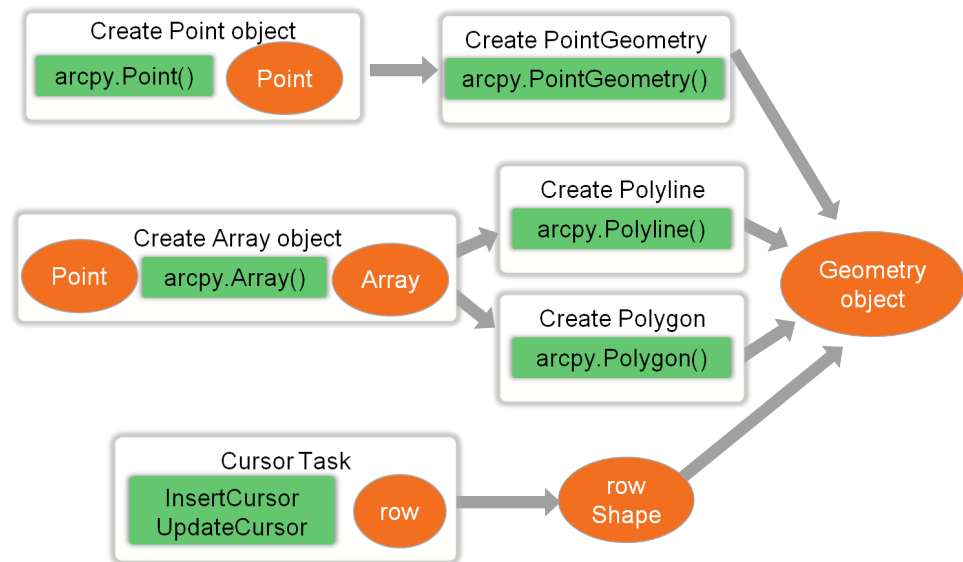
- Access the ArcPy classes to create geometry objects.
- Use the geometry object with the arcpy.da cursors.
- Obtain the geometry object from the Shape field and use it in a geoprocessing tool.
- Populate a new feature class with an arcpy.da cursor using a Python list of coordinates.

## Creating geometry objects

When performing geoprocessing tasks, you may need to run a tool that uses coordinate or geometry information for input. In a normal workflow, you might create a temporary feature class to hold the geometry, populate the feature class with an Insert cursor, and then use the feature class as input to the geoprocessing tool.

An alternative to storing the geometry in a temporary feature class is to use the ArcPy Geometry classes.

**Figure 6.1**  
Workflows for creating  
geometry objects.



Geometry objects can be created in memory from the Geometry, MultiPoint, PointGeometry, Polyline and Polygon ArcPy classes. The geometry object can then be populated to represent a Shape and then used for input in a Geoprocessing tool (removing the need to store the geometry in a temporary feature class). Using a **geometry object** can make your workflows simpler than they would be if you were to use a Feature Layer or Feature Class as input.

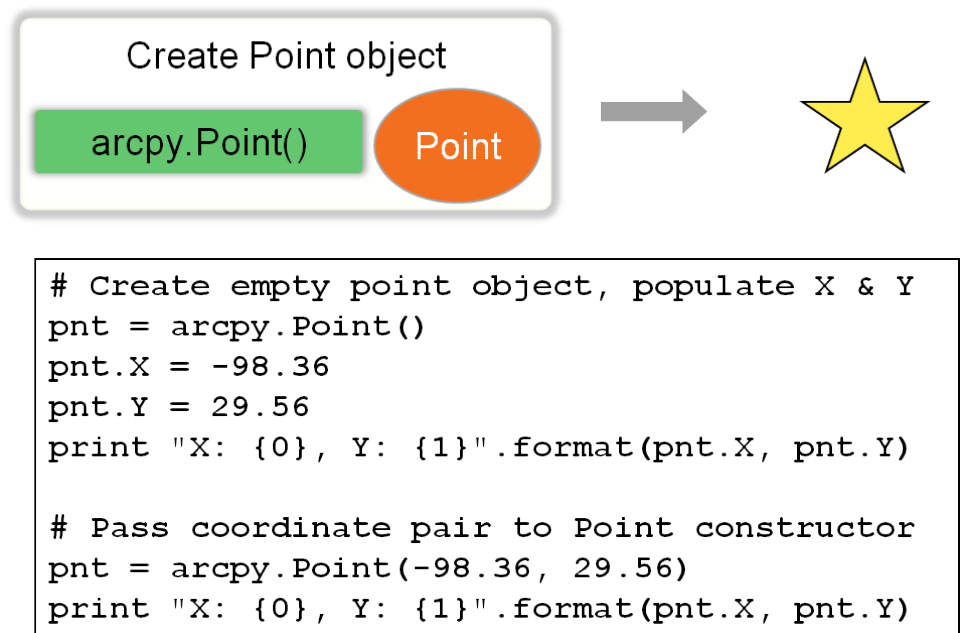


## Creating Point objects

The Point object is typically used in workflows to store coordinate pairs of X & Y values, and is used to construct PointGeometry, Polyline, Polygon, and MultiPoint geometry objects.

When accessing a point feature class, each point feature returned by a cursor is a PointGeometry object. For all other feature types, such as MultiPoint, Polyline, and Polygon, equivalent geometry objects are returned. If the feature contains multiple parts, then an array containing multiple arrays of point objects is returned—one array is returned for each part of the feature.

**Figure 6.2**  
Creating a Point object.



Syntax:

```
arcpy.Point ({X},{Y},{Z},{M},{ID})
```

Table 6.1

The arcpy.Point parameters

| Parameter | Description                   |
|-----------|-------------------------------|
| X         | The X Coordinate of the point |
| Y         | The Y coordinate of the point |
| Z         | The Z coordinate of the point |
| M         | The M coordinate of the point |
| ID        | The shape ID of the point     |



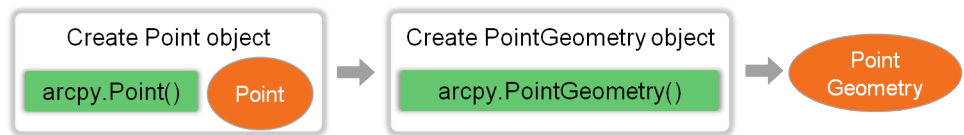
ArcGIS 10.1 Help Library:  
*PointGeometry*

The Point object does not support a SpatialReference property. The SpatialReference property is used for spatial comparisons, creating new point features in a feature class, and interpretation of provided x,y coordinate values. To specify this property, create a PointGeometry object from the Point object and specify the SpatialReference.

## The PointGeometry object

The Point Geometry object is a shape that has no area or length and supports all geometry object properties, including a spatial reference. It is constructed from a Point object. A typical workflow for creating a PointGeometry is to first create and populate a Point object, and then pass the Point object to the `arcpy.PointGeometry()` as the input parameter.

**Figure 6.3**  
Creating a PointGeometry  
object.



The populated PointGeometry object can then be copied to a new feature class with the `arcpy.CopyFeatures()` tool. It can be passed to an `UpdateCursor` or `InsertCursor` and assigned to the index position of the Shape field in the current row of the cursor.

The PointGeometry object supports methods for comparing to other geometries. A typical scenario would be to create the PointGeometry object, then access the Shape geometry from a cursor. The PointGeometry methods for performing a comparison to the Shape geometry include `contains`, `disjoint`, `equals`, `within`, and `overlaps`. A boolean `True` or `False` will be returned.

Syntax:

```
arcpy.PointGeometry(inputs, {spatial_reference},
                    {has_z}, {has_m})
```

Table 6.2

arcpy.PointGeometry parameters

| Parameter         | Description                                                                                                                                         |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| inputs            | Coordinates used to create the object. Parameter datatype can be a Point or Array object.                                                           |
| spatial_reference | Spatial reference of the new geometry. Use arcpy.SpatialReference to create a new object or arcpy.Describe to obtain an existing spatial reference. |
| has_z             | True if Z is enabled for the geometry; False if not enabled.                                                                                        |
| has_m             | True if M is enabled for the geometry; False if not enabled.                                                                                        |

## Creating Polyline geometry objects

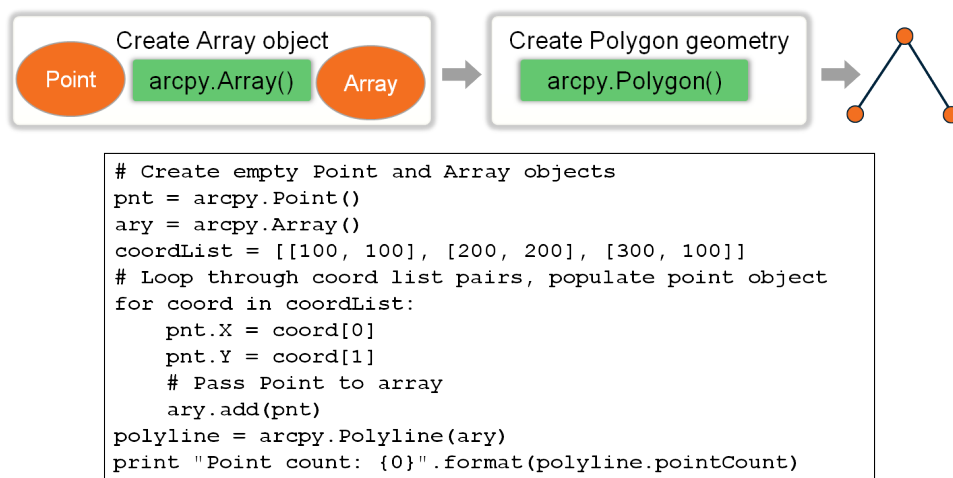
A Polyline is a series of connected segments that form a path. A Polyline geometry object can be constructed from an array of Point objects, which define each segment in the path.

The Polyline geometry object can then be used as input to geoprocessing tools, used for comparison with other geometry, or can be passed to the Shape field with a cursor for updating a feature or creating a new feature.

A typical workflow for creating a new Polyline geometry object is to:

- Create an empty Point and Array object.
- Populate each Point with coordinate pairs, and add the Point to the Array object.
- Loop through the remaining coordinate pairs.
- Pass the Array object to the `arcpy.Polyline` class constructor.

**Figure 6.4**  
Creating a Polyline object.



Syntax:

```
arcpy.Polyline(inputs, {spatial_reference},
               {has_z}, {has_m})
```

Table 6.3

arcpy.Polyline parameters

| Parameter         | Description                                                                                                                                         |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| inputs            | Coordinates used to create the object. Parameter datatype can be a Point of Array object.                                                           |
| spatial_reference | Spatial reference of the new geometry. Use arcpy.SpatialReference to create a new object or arcpy.Describe to obtain an existing spatial reference. |
| has_z             | True if Z is enabled for the geometry; False if not enabled.                                                                                        |
| has_m             | True if M is enabled for the geometry; False if not enabled.                                                                                        |



ArcGIS 10.1 Help Library:  
*Polyline (arcpy)*

The Polyline geometry object supports methods for comparing to other geometries. A typical scenario would be to create the Polyline geometry object, then access the Shape geometry from a cursor. The Polyline methods for performing a comparison to the Shape geometry include contains, crosses, disjoint, equals, touches, within, and overlaps. A boolean True or False will be returned.

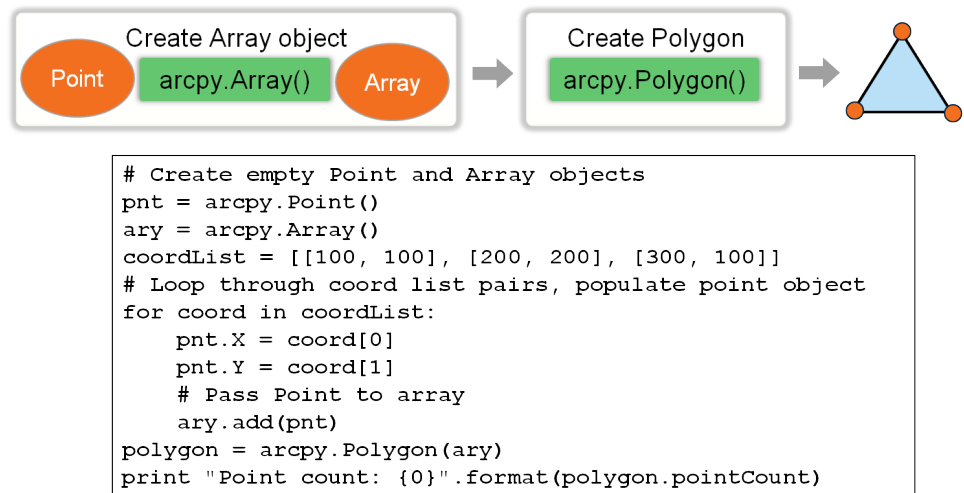
## Creating Polygon geometry objects

A Polygon is defined by a series of connected segments which form a closed path. A Polygon geometry object can be constructed from an array of Point objects, which define each segment in the path. Each Point object contains an X,Y coordinate pair.

Constructing a Polygon geometry object is very similar to constructing a Polyline geometry object. A typical workflow is:

- Create an empty Point and Array object.
- Populate each Point with coordinate pairs, and add the Point to the Array object.
- Loop through the remaining coordinate pairs.
- Pass the Array object to the `arcpy.Polygon` class constructor.

**Figure 6.5**  
Creating a Polygon object.



The `arcpy.Polygon` constructor will always close the Polygon geometry object, so there is no need to add the first point back into the array to close the polygon.

Syntax:

```
arcpy.Polygon(inputs, {spatial_reference},
               {has_z}, {has_m})
```

Table 6.4

arcpy.Polygon parameters

| Parameter         | Description                                                                                                                                          |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| inputs            | Coordinates used to create the object. Parameter datatype can be a Point or Array object.                                                            |
| spatial_reference | Spatial reference of the new geometry. Use arcpy.SpatialReference to create a new object or arcpy.Describe to obtain an existing spatial reference.. |
| has_z             | True if Z is enabled for the geometry; False if not enabled.                                                                                         |
| has_m             | True if M is enabled for the geometry; False if not enabled.                                                                                         |



ArcGIS 10.1 Help Library:  
*Polygon (arcpy)*

The Polygon geometry object supports methods for comparing to other geometries. A typical scenario would be to create the Polygon geometry object, then access the Shape geometry from a cursor. The Polygon methods for performing a comparison to the Shape geometry include contains, crosses, disjoint, equals, touches, within, and overlaps. A boolean True or False will be returned.





ArcGIS 10.1 Help Library:  
*Geometry (arcpy)*

## The geometry object

Geometry objects define a geometric shape and spatial location. A geometry object can be created from the `arcpy.Geometry` class constructor or by using the `ArcPy` `Point`, `MultiPoint`, `PointGeometry`, `Polyline`, or `Polygon` class constructors.

Geometry objects can be used for spatial comparisons with features, to update or create new features when used in conjunction with an `arcpy` cursor, and for input to and output from geoprocessing tools.

### Syntax:

```
arcpy.Geometry(geometry, inputs, {spatial_reference},
               {has_z}, {has_m})
```

#### **arcpy.Geometry parameters**

Table 6.5

| Parameter         | Description                                                                                                                                                                       |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| geometry          | The geometry object type <ul style="list-style-type: none"> <li>• point</li> <li>• multipoint</li> <li>• polyline</li> <li>• polygon</li> </ul>                                   |
| inputs            | The coordinate pairs used to create the geometry object; the datatype can be either <code>Point</code> or <code>Array</code> .                                                    |
| spatial_reference | The spatial reference of the new geometry. Use <code>arcpy.SpatialReference</code> to create a new object or <code>arcpy.Describe</code> to obtain an existing spatial reference. |
| has_z             | True if Z is enabled for the geometry; False if not enabled.                                                                                                                      |
| has_m             | True if M is enabled for the geometry; False if not enabled.                                                                                                                      |

If the geometry parameter is set to point, the `arcpy.Geometry` class constructor will ignore the `spatial_reference` parameter.

When accessing the Shape geometry from a cursor, you can assign the Shape geometry to an empty Geometry object and then access the geometry properties.

For example, you may wish to sum the length of all the polylines in a feature class. Using a `SearchCursor`, you could loop through each feature in the feature class, assign the shape to an empty geometry object, and then access the geometry object's length property. Once finished processing the features returned by the cursor, you could then report the total length.

An alternative to accessing each feature's length with an `arcpy.da SearchCursor` is to create a **geometry list**.

### Example:

```
import arcpy
arcpy.env.workspace = "C:/MyData/MyProject.gdb"
g = arcpy.Geometry()
geomList = arcpy.CopyFeatures_management(
    "Railroads", g)

length = 0
for geom in geomList:
    length += geom.length
print "Total length in miles: {}".format(
    round(length / 5280, 2))
```

## Constructing multipart geometry

Workflow for creating standard, single-part geometries:

1. Create a Point object.
2. Populate the values to store the coordinate pair.
3. Add it to an Array.

This process repeats until all of the coordinate pairs for the feature are stored in the Array. The Array is then passed to the MultiPoint, PointGeometry, Polyline, or Polygon class constructor to create the geometry object.

A multipart feature can be constructed from an array or an array of points. Each array of points defines a geometry part of the feature. If you were to create a two-part polyline, you would create two arrays—one for each geometry part. Once the arrays are loaded with the Point objects to define each part, the two arrays are then passed as a list to a third array. The third array is then passed as the inputs parameter to the geometry class constructor.

### Example:

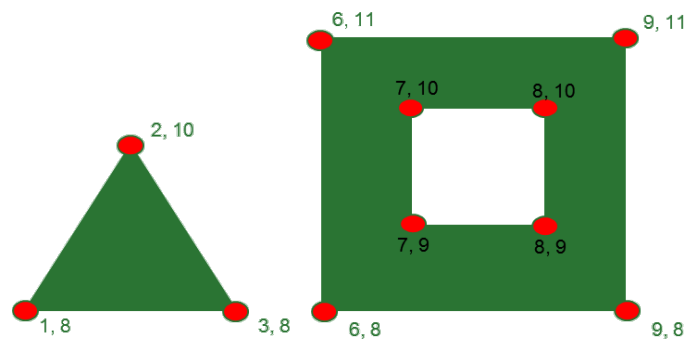
```
partOne = arcpy.Array([arcpy.Point(5997624.6225,
                                   2069868.8208),
                       arcpy.Point(5997674.94199,
                                   2069833.81741)])
partTwo = arcpy.Array([arcpy.Point(5997616.44497,
                                   2069862.32774),
                       arcpy.Point(5997670.57373,
                                   2069824.67456)])
ary = arcpy.Array([partOne, partTwo])
multiPartPolyline = arcpy.Polyline(ary)
print multiPartPolyline.partCount
```

## Constructing multipart polygons

When constructing multipart polygon geometry objects, the series of connected points that define the boundary of the polygon is called a *ring*. Multipart polygons are constructed from a series of rings, which can be either interior or exterior rings. Interior rings are constructed in a counterclockwise manner, and exterior rings are constructed in a clockwise manner.

The following graphic displays a single polygon feature with two parts. To create this feature, you would need an array of points to define the interior ring on the right-hand shape, an array of points to define the exterior ring, and a third array to define the exterior ring on the left shape. The three arrays would be added to a forth array as a list, and then passed to the inputs parameter for the `arcpy.Polygon` class.

**Figure 6.6**  
Constructing a multipart  
polygon geometry object.



### Code to create the multipart polygon geometry object:

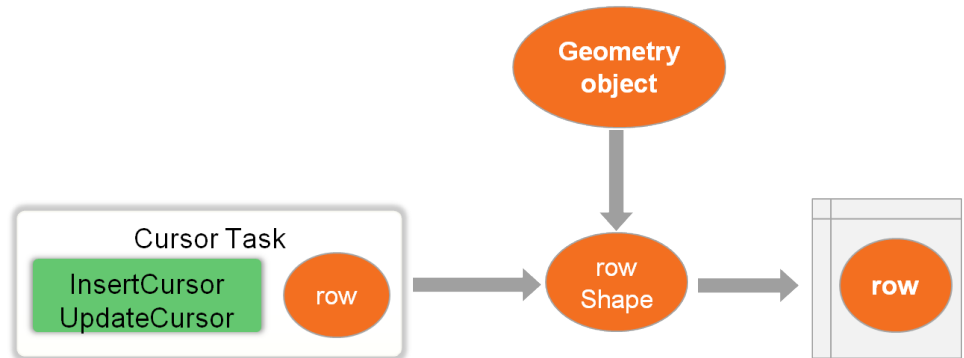
```
# In this example, there are two parts to the
# polygon with one part containing an interior ring.
intRingAry = arcpy.Array([arcpy.Point(7,10),
                           arcpy.Point(7,9),
                           arcpy.Point(8,9),
                           arcpy.Point(8,10)])
extRingAry = arcpy.Array([arcpy.Point(9,11),
                           arcpy.Point(9,8),
                           arcpy.Point(6,8),
                           arcpy.Point(6,11)])
extRingAry2 = arcpy.Array([arcpy.Point(2,10),
                            arcpy.Point(3,8),
                            arcpy.Point(1,8)])
array = arcpy.Array([extRingAry, intRingAry,
                     extRingAry2])
poly = arcpy.Polygon(array)
print "Parts: {0}, points: {1}".format(
    poly.partCount,
    poly.pointCount)
```

## Creating and updating feature geometry

Updating features in an existing feature class or inserting new features into a feature class is accomplished using the `arcpy.da` cursors. The cursor can access the feature geometry by working with the Shape field in the current cursor row. When accessing feature geometry or passing a new geometry object to the Shape field, use the `SHAPE@` tokens in the cursor field list for more efficient cursor performance.

**Figure 6.7**

An `UpdateCursor` or `InsertCursor` is used to pass geometry to Shape field.



### Code examples:

```
# Writing geometries with an arcpy.da.InsertCursor
cursor = arcpy.da.InsertCursor(featClass, "SHAPE@")
ary = arcpy.Array([arcpy.Point(9,11),
                   arcpy.Point(9,8),
                   arcpy.Point(6,8),
                   arcpy.Point(6, 11)])
polygon = arcpy.Polygon(array)
cursor.insertRow([polygon])
del cursor

# Updating geometries with an arcpy.da.UpdateCursor
exp = "" "Name" = 'Balboa Park' ""
cursor = arcpy.da.UpdateCursor(featClass2,
                               "SHAPE@XY", exp)

for row in cursor:
    row[0] = arcpy.Point(6285430.0, 1844965.66)
    cursor.updateRow(row)
del cursor

# Reading geometries with a arcpy.da.SearchCursor
for row in arcpy.da.SearchCursor("Parks",
                                  ["SHAPE@AREA", "NAME"])
    area = row[0]
    acreage = area / 43560
    print("{0} Acres: {1}, Area: [2]".format(
        row[1], acreage, row[0])
```



ArcGIS 10.1 Help Library:  
Using in-memory  
workspace

## Using geometry object with geoprocessing tool

Many geoprocessing tools will accept geometry for input and can also output geometry as the tool's result object. You can then use the result object as input to other geoprocessing tools. For example, you could create a Point object, pass it to the Buffer tool as input, take the output from the Buffer tool and pass it to the Clip tool as the clipping polygon.

ArcGIS provides an in-memory workspace where geoprocessing tool output can be stored. This provides an alternative to writing output to a folder or geodatabase. (Writing results to the in-memory workspace is typically much faster than writing to disk.)

To write to the in-memory workspace, specify the path `in_memory`, as in `in_memory/featClass`. Any data written to the in-memory workspace will be deleted when the application is closed. The `arcpy.Delete_management()` tool can also be used to delete data in the in-memory workspace.

### Code example:

```
# Use an in memory workspace to store
# geoprocessing tool output as intermediate data.
# Use arcpy.da SearchCursor to access the in_memory
# features
import arcpy
pnt = arcpy.Point(6282633.845, 1838254.582)

# Create geometry object from Point
geom = arcpy.Geometry("Point", pnt)

# Buffer the point and use in_memory
# feature class for output
bufferPnt = arcpy.Buffer_analysis(geom,
                                   "in_memory/bufferPnt", "5280 feet")

# Clip meters based on in_memory buffer output
clipMeters = arcpy.Clip_analysis("ParkingMeters",
                                   bufferPnt, "in_memory/clipMeters")

# Access in_memory\clipMeters features using
# an arcpy.da SearchCursor and report the
# meter ID and meter type
with arcpy.da.SearchCursor("in_memory/clipMeters",
                           ["MeterID", "TYPE"]) as cursor:
    for row in cursor:
        print "ID: {0}, Type: {1}".format(row[0],
   row[1])
```

## Accessing geometry objects

### Workflow:

1. Import the ArcPy site package and set the current workspace.
2. Assign feature class names to variables.
3. Use a for-in loop on a SearchCursor to access the Shape X,Y and NAME field values, where the NAME field contains a value.
4. Report the feature name value and X,Y values.
5. Use a SearchCursor in a with loop to access and report the ID and SHAPE@AREA values.
6. Use an UpdateCursor to update the Shape X,Y for an existing feature.
7. Use an empty Geometry object in conjunction with the arcpy.CopyFeatures tool to output a geometry list containing Shape geometry objects. Report the total geometry object length property values.

### Best practices for working with geometry objects and cursors

- Use in\_memory workspaces for more efficient processing. The outputs from geoprocessing tools can be stored in the in\_memory workspace and removes the need to write intermediate data to disk.
- Use a geometry list when summarizing Shape attributes. The arcpy.CopyFeatures\_management tool is an example where the output from a geoprocessing tool can be passed to a geometry list. A simple for-in loop can then be written to access each geometry object in the list to summarize the geometry object property values.
- Use the SHAPE@ geometry tokens in an arcpy.da cursor as a shortcut rather than accessing the full Shape geometry object. The cursor performance will be much higher than if the Shape field is specified in the arcpy.da cursor field list parameter.
- Use a Python with statement when working with cursors and feature geometry. When Python exits the with statement, the data source and cursor will be automatically closed and any locks on the data will be released. To navigate the cursor again, use the cursor.reset() method to reset the cursor iteration.





50 minutes



## Exercise 6: Work with geometry objects and cursors

Once the geometry object has been created, it can be used to create new features or modify existing features in feature classes. It can also be passed to a geoprocessing tool for both input and output parameters. You will create new Point and Array objects, create new Polyline and Polygon geometry objects, work with geometry object properties, and use cursors in conjunction with geometry objects to report values, update features, and create new features.

In this exercise, you will:

- Create Point, Array, Polyline, and Polygon objects.
- Report on geometry object properties.
- Use an `arcpy.da.SearchCursor` to return Shape geometry properties.
- Use an `arcpy.da.UpdateCursor` to modify feature location.
- Use an `arcpy.da.InsertCursor` to create and populate new feature.
- Optionally use a geometry object with a geoprocessing tool to clip features.

Exercise workflow:

- Create Point and Array objects.
- Create Polyline and Polygon geometry objects.
- Report on Shape geometry properties.
- Use an `arcpy.da.UpdateCursor` and a Point object to modify the X,Y for an existing feature.
- Use an `arcpy.da.InsertCursor` and a Polyline geometry object to create a new feature.
- Use a Polyline geometry object as an input to a geoprocessing tool to extract features to a new feature class.

## Step 1: Create geometry objects

In this step, you will create a Point object, a Polyline geometry object, and Polygon geometry object. First you will create a new Point object and examine the object properties.

- a** In the ArcGIS 10.1 Help Library, search for *Point (arcpy)*.



Alternatively, navigate to Geoprocessing > ArcPy > ArcPy classes > Geometry > Point.

- b** Review the following help topic sections:

- Summary
- Discussion
- Syntax
- Code Sample—Point example

- c** Examine the Syntax section to answer the following questions:

1. At minimum, what is required to create a populated Point object?

---

2. What code would be required to create a new Point object, where X is 2000 and Y is 1000, assigning the Point object to a variable named `pnt`?

---

- d** Start PythonWin and create a new script named **MyCreateGeomObjects.py** in the C:\Student\PYTH\Geometry\_objects folder.

- e** Using your answer to the second question, write code to create a new populated Point object and report the Point object's X,Y coordinates to the Interactive Window.

- f** Check the code syntax and run the script.

- g** Verify that the coordinates displayed in the Interactive Window match the X,Y coordinate pair used to create the Point object.

Your next task is to create a new Polyline and Polygon geometry object.

- h** Return to the ArcGIS 10.1 Help Library, and review the topics for the Polyline and Polygon geometry classes.

*Hint:* Search for **polyline (arcpy)** and **polygon (arcpy)** or navigate to:

- Geoprocessing
- ArcPy
- ArcPy classes
- Geometry
- polyline/polygon

3. What objects must be created in order to populate a new Polyline or Polygon geometry object?
- 

- i** Use the answer to the above question to write code creating a new Polyline geometry object:

- Create an empty Point object and an empty Array object. Assign the Point object to a variable named **pnt** and the Array object to a variable name **ary**.
- Create a Python list containing the following X,Y coordinate pairs and assign it to a variable named **coords**.

| X-coordinate | Y-coordinate |
|--------------|--------------|
| 100          | 200          |
| 200          | 400          |
| 300          | 700          |
| 600          | 800          |
| 500          | 700          |

- Use a for-in loop to iterate through the list of coordinate pairs:
  - Populate the Point object X & Y properties from each coordinate pair in the list.
  - Use the add method on the Array object to add the Point.
- Outside the for-in loop, pass the Array object to the Polyline class constructor. Assign the Polyline geometry object to a variable named **polyLine**.



If you need additional assistance or are stuck on an exercise step, you can refer to the solution scripts in the C:\Student\PYTH\Geometry\_objects\Solution folder.

- j Check for syntax errors and run your script.
- k In the PythonWin Interactive Window, write the following line of code to report the **pointCount** property of the Polyline geometry object:  

```
print "Number of points: {0}".format(polyLine.pointCount)
```

4. What is the value reported in the Interactive Window?

---

- l Modify the script to create a new Polygon geometry object.  
*Hint:* There is only one line of code that needs to be changed in the script.

- m Check for syntax errors and run your script.

5. What is the value of the pointCount property for the Polygon geometry object?

---

- n Close your script, but keep PythonWin open.

## Step 2: Access Shape geometry

In this step, you will use an `arcpy.da.SearchCursor` to iterate through a feature class and report the Shape properties for each feature.

- a In PythonWin, create a new script named **MyReportShapeGeometry.py** in the C:\Student\PYTH\Geometry\_objects folder.

- b** Using the following workflow steps as a guide to writing a script that uses a SearchCursor to report Shape geometry properties:

- Import the ArcPy site package.
- Set the current workspace environment to `C:/Student/PYTH/Geometry_objects/SanDiego.gdb`.
- Create a **for-in** loop on the `arcpy.da.SearchCursor`, passing these parameters:
  - `"Freeways"` for the `in_table` parameter
  - `["SHAPE@XY", "OID@", "SHAPE@LENGTH"]` for the `field_names` parameter
- Assign the return from the SearchCursor to a variable named `row`.
- Inside the loop, report the **X**, **Y**, **ID**, and **length** properties of the `row` object to the Interactive Window.



The SHAPE@XY token returns the X,Y values as a tuple. You can slice the row object to obtain the values (as in `row[0][0]` for the **X** value and `row[0][1]` for the **Y** value).

- c** Check your script for syntax errors and then run the script.
- d** Close the script, but keep PythonWin open.

### Step 3: Update existing features

In this step, you will work with an `arcpy.da.UpdateCursor` to modify the spatial location of an existing feature in the MajorAttractions feature class.

- a** Start ArcMap and open the SanDiegoMarina.mxd located in the C:\Student\PYTH\Geometry\_objects folder.
- b** From the Bookmarks menu, select Balboa Park, and note the location of the Balboa Park feature in the MajorAttractions layer.
- c** In PythonWin, create a new script named **MyUpdateCreateFeatures.py** in the C:\Student\PYTH\Geometry\_objects folder.

- d Using the following workflow steps as a guide to writing a script that uses an `arcpy.da.UpdateCursor` to modify the X,Y location of Balboa Park:

- Import the ArcPy site package.
- Set the current workspace environment to  
`C:/Student/PYTH/Geometry_objects/SanDiego.gdb`
- Assign the following variables:
  - `featClass = "MajorAttractions"`
  - `fields = ["SHAPE@XY"]`
  - `exp = """ "NAME" = 'BALBOA PARK' """`
- Create an empty Point object and assign it to a variable named `pnt`.
- Modify the `pnt` properties to the following:
  - `pnt.X = 6285430.0`
  - `pnt.Y = 1844965.66`
- Create a **with** loop on the `arcpy.da.UpdateCursor`, passing these parameters to the `UpdateCursor`:
  - `featClass` for the `in_table` parameter
  - `fields` for the `field_names` parameter
  - `exp` for the `where_clause` parameter
- Assign the cursor to a variable named `cur`.



Refer to ArcGIS 10.1 for Help Library and search for **UpdateCursor** to view code samples. These code samples contain examples of creating an `arcpy.da.UpdateCursor` using a `with` statement.

- Inside the **with** statement, create a **for-in** loop on `cur`. Assign the rows returned from `cur` to a variable named `row`.
- Within the `for-in` loop:
  - Assign the Point object to `row[0]`
  - Call the `updateRow()` method on `cur`, passing `row` as the parameter.

- e Check for syntax errors and run the script.
- f Leave PythonWin open, and open ArcMap.
- g In ArcMap, from the View menu, select Refresh.

Notice the updated location for the Balboa Park feature.

- h Leave ArcMap open and return to PythonWin.

## Step 4: Create new features

In this step, you will write code to create a new feature in the Freeways feature class. In your script, you will use Point and Array objects, along with a Python List of coordinates to create a new Polyline geometry object. Working with an `arcpy.da.InsertCursor`, you will insert the Polyline into the Freeways feature class.

- a In PythonWin, comment the code in your script below the line of code that creates the empty Point object.

- b** Using the following workflow steps as a guide for adding code to your script that will use an **arcpy.da.InsertCursor** to add a new Polyline to the Freeways feature class:

- Assign the following variables:
  - **featClass2** = "Freeways"
  - **fields2** = ["SHAPE@", "STREET\_NAM"]
  - **fldVal** = "Balboa Park Drive"
- Create a new **Array** object and assign it to **ary**.
- Using your skills from the first step in the exercise, create a new **Polyline** geometry object with the following coordinates, in the form of:
 

```
coords = [[<coord1X>, <coord1Y>], [<coord2X>, <coord2Y>],
          [<coord3X>, <coord3Y>], [<coord4X>, <coord4Y>]]
```

| X           | Y           |
|-------------|-------------|
| 6284696.620 | 1844282.464 |
| 6284739.145 | 1844632.515 |
| 6284919.091 | 1844881.011 |
| 6285184.726 | 1845026.683 |
| 6285423.068 | 1845060.988 |

- Assign the object returned from the **arcpy.Polyline** class to a variable named **polyLine**. Pass **ary** to the **arcpy.Polyline** parameter.
- Create a new **arcpy.da.InsertCursor** and assign the cursor to a variable named **cur2**. Pass **featClass2** and **fields2** to the cursor for the two parameters.
- Call the **insertRow** method on the **cursor**, passing **polyLine** and **fldVal** as a list to the method's parameter.
- As the final line of code in the script, **del cur2** to remove the exclusive lock on the Freeways feature class.



If you need additional assistance or are stuck on an exercise step, you can refer to the solution scripts in the C:\Student\PYTH\Geometry\_objects\Solution folder.

- c** Debug and run your script.
- d** Return to the SanDiegoMarina.mxd in ArcMap, and navigate to the New Freeway bookmark.



- e Confirm that the new feature has been added to the Freeways layer and that the STREET\_NAM value is 'Balboa Park Drive'.



You may need to refresh the view in order to see the new feature.

- f Close ArcMap.
- g If you have time and would like to work on the optional step, keep PythonWin open. Otherwise, close PythonWin.

## Step 5: (Optional) Use geometry object with geoprocessing tool

This is an optional step and is not included in the estimated exercise time.

In this step, you will work with the Corvallis geodatabase, a Polygon geometry object, and a Geoprocessing tool to clip out the Downtown streets and store them in a separate feature class.

- a In PythonWin, create a new script named **ExtractDowntownStreets.py** in the C:\Student\PYTH\Geometry\_objects folder.
- b Import **arcpy** and set the current workspace to **C:/Student/PYTH/Geometry\_objects/Corvallis.gdb**.
- c Create a new Point and Array object.
- d Populate a polygon with the following X,Y coordinates:

| X-coordinate | Y-coordinate |
|--------------|--------------|
| 1277000.0    | 344000.0     |
| 1283000.0    | 344000.0     |
| 1283000.0    | 336000.0     |
| 1277000.0    | 336000.0     |

- e Using the correct Clip geoprocessing tool, clip the **StPaved** feature class against the Polygon geometry object, creating the new **DowntownStreets** feature class.



**Be sure to choose the correct Clip tool—one of the Clip tools only works with rasters.**

- f Check the script syntax and run the script.

Your next task is to verify that the script ran successfully.

- g In ArcMap, open the CorvallisStreets.mxd within the C:\Student\PYTH\Geometry\_objects folder.
- h Navigate to the Downtown bookmark.
- i Add the DowntownStreets feature class to the map and choose symbology that will contrast the downtown streets from the Streets layer.

Your final task is to run the script in the Python window.

- j In the table of contents, right-click the DowntownStreets layer and select Remove.
- k From the Geoprocessing menu, choose Geoprocessing Options.
- l Confirm that the check box to Overwrite the output of geoprocessing operations is selected.
- m Open the Python window.
- n Load the script into the Python window, then press Enter twice to run the script.
- o Save your changes to ArcMap, then close ArcMap.
- p Close PythonWin.

In the exercise, you created Point and Array objects, and Polyline and Polygon geometry objects. You accessed Shape geometry properties using a SearchCursor, updated the Shape X,Y for an existing feature using an UpdateCursor, created new features using an InsertCursor, and passed a geometry object to a geoprocessing tool to extract features to a new feature class.

## Lesson review

1. To create a Polyline or Polygon geometry object, what two objects must first be created?

---

---

2. List some geometry tokens that efficiently access Shape geometry properties.

---

---

3. Provide a couple of examples in which you would use a geometry object with a geoprocessing tool.

---

---

---

## Answers to Lesson 6 questions

### Exercise 6: Work with geometry objects and cursors (page 6-19)

1. At minimum, what is required to create a populated Point object?

**X, Y coordinates**

2. What code would be required to create a new Point object, where X is 2000 and Y is 1000, assigning the Point object to a variable named `pnt`?

**`pnt = arcpy.Point(2000, 1000)`**

3. What objects must be created in order to populate a new Polyline or Polygon geometry object?

**A Point object and an Array object. (A Python List of X,Y coordinate pairs is optional.)**

4. What is the value reported in the Interactive Window?

**The value reported in the Interactive Window is 5.**

5. What is the value of the `pointCount` property for the Polygon geometry object?

**The value of the `pointCount` property for the Polygon geometry object is 6.**

# 7

## Sharing scripts

### Key terms

model

parameter

script

script tool

system tool

### Introduction

Up to this point, you have been writing scripts that are using hard-coded paths for data and variables with hard-coded values. This works fine in scripts that may be used for one-off workflows or for when a script operates on the same set of data from time to time.

There will be times when you would like to share a script with another user, run a script on data in a different location, or even work through a workflow where the input and output parameters would change for each scenario.

By adding arguments to your scripts, you can make the script more dynamic, better fit your automation and analysis workflows, and shareable through Geoprocessing packages and services.

### Topics covered

- Passing parameters to scripts
- Attaching a script to a custom tool
- Running a script in Python window
- Running a script from Results window
- Sharing a script as a geoprocessing package

### Learning objectives

After completing this lesson, you will be able to:

- Create a custom script tool.
- Run the custom script tool in the Python window.
- Share the results as a geoprocessing package.

Terms commonly used when sharing scripts

Table 7.1

| Term               | Definition                                                                                                                                                                                                                                                                         |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>model</b>       | Workflows that run a sequence of geoprocessing tools. The output of a tool can be used as input to the next tool in the sequence.                                                                                                                                                  |
| <b>parameter</b>   | Values that can be passed to the script, or output from the script. The term <i>parameter</i> and <i>argument</i> can be used interchangeably.                                                                                                                                     |
| <b>script</b>      | A text file (.py) containing Python statements and functions that can be executed. The script can branch based upon values and can loop through sequences of values. A script using the ArcPy module will contain one or more geoprocessing tools that are executed in a sequence. |
| <b>script tool</b> | A Python script that is added to a custom toolbox, thus making it available to be run within ArcGIS.                                                                                                                                                                               |
| <b>system tool</b> | A geoprocessing tool that is provided with the installation of ArcGIS. The over 850 system tools are organized by functionality into toolboxes and further grouped into toolsets.                                                                                                  |

## Scripting advantages in ArcMap

### *What are some advantages gained by running your script in ArcMap?*



Running a script outside of ArcGIS means it could be run from the operating system prompt, from the task scheduler, or from your favorite Python IDE.



Script tools can also be run from within a model in ModelBuilder.

Geoprocessing tools typically perform a single task on geographic data. When used in ModelBuilder or in a Python script, you execute tools as a sequence of tasks, with the output from a tool used as input to the next tool. This sequence of tools can be run within ArcGIS, or they can be run outside ArcGIS if they are accessed from a script.

Running a script within ArcGIS can be accomplished by creating a script tool or by loading a script into the Python window. When creating a script tool, you can define the script parameters, and then open and execute the tool from the tool dialog. The tool dialog will check each script parameter value and will run only when all of the required parameters are valid.

Script tools can also be run from the Python window. To run script tools in the Python window, import the custom toolbox and then access the script tool. If the script tool is run inside the ArcMap application, the script tool dialog is aware of the layers in the map.

## Making scripts dynamic

Scripts can be considered *static* or *dynamic*. When a script is *static*, all paths and values are hard-coded. This does not allow for any change to the values when the script is run. When a script is *dynamic*, the user running the script can specify the values for the paths and variables at runtime. This makes the script more flexible to changes in values. It also allows for running scenarios and 'what-ifs'.

When scripts contain hard-coded values, such as feature class names or paths to data, the script knows exactly where to go to access the data and work with the defined values. This can be ideal for one-off projects or when the script is routinely processing a set of data.

What if you wanted to run the script multiple times, accessing a different set of data or path each time? Or what if you wanted the end user to select the feature class or geodatabase before running the script?

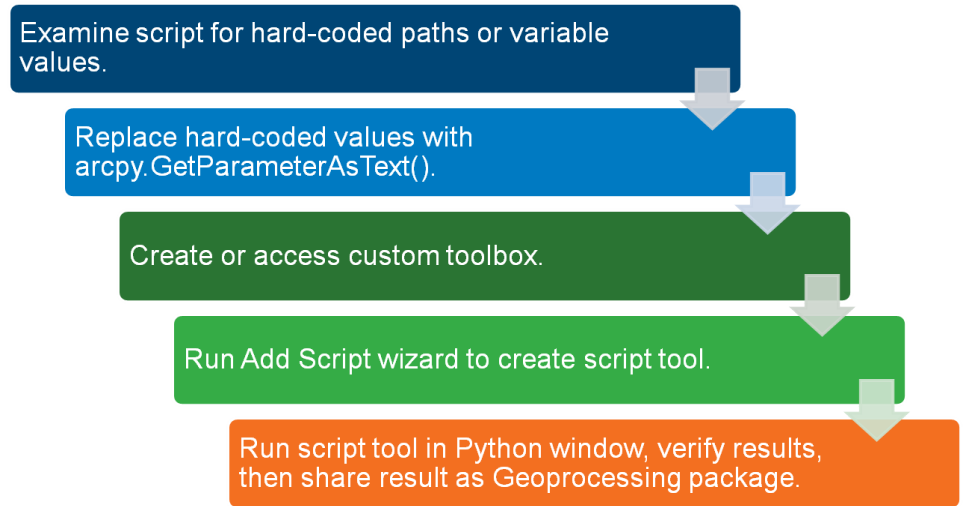
The **`arcpy.GetParameterAsText()`** function allows you to pass values to the script at runtime. If your script is going to be used in a model, the **`arcpy.SetParameterAsText()`** function will pass values to the model as output. Using the script output is a way that you can tie the results of your script to the next tool in the model.

Once your script is dynamic, you can create a script tool that runs your script. The script tool can prompt the user to provide arguments, which are passed from the script tool dialog to your script at runtime. The dialog also validates your entries for a match with each parameter defined for the script tool, and will run only after all parameters are valid.



## Creating and sharing a script tool

**Figure 7.1**  
Workflow for creating and  
sharing a script tool.





## Advantages of attaching a script to a custom tool

### The script becomes part of the geoprocessing framework.

- You can run the script in a model, from the Python window, and from the Results window.
- Your script tool will honor the current geoprocessing environment settings and the user will be able to access standard help documentation for that tool (e.g., scripting syntax, usage).

### The script receives a user-friendly interface.

- When users enter a name, such as a feature class or table, they do not need to type in the full path—they can use the Browse button.
- If users want to build a SQL statement, the Query Builder button can be used. This provides a valid SQL statement to the parameter.

### The custom tool dialog box prevents many potential errors.

- If a user enters the name of an input feature class that does not exist, or a name for an output feature class that does already exist, the dialog will notify the user.
- When the parameter requires a certain type of data (e.g., a feature class), the dialog box will only allow the user to enter values for that type of data. Valid values for a parameter can be obtained from other parameters, such as listing the fields for a feature class. Filters can provide a set of values for a parameter, such as a range of values or a list of codes.



System toolboxes are read-only, so a custom toolbox must be created before a custom tool can be created.

## Using the Add Script wizard

This easy-to-use wizard attaches a script to a custom tool in a custom toolbox. A custom toolbox can live in a folder or a geodatabase. There are three panels in the Add Script wizard.

### First panel

- Provide the script tool's internal name, display name, and a description that displays in the tool dialog.
- *Optionally* allow the tool to honor the Background Processing geoprocessing environment setting.
- *Optionally* allow the tool to store the path to the script as a relative path.

**Figure 7.2**  
First panel of the Add Script wizard.

**Add Script**

Name:  
FeaturesToFeatures

Label:  
Feature Class to Feature Class

Description:  
Copies feature classes from one location to another location. Shapefiles and geodatabase feature classes are valid inputs for this tool. The tool will allow for an option SQL statement to be applied to the input data, along with a filter for specifying the fields that will not be included in the tool output.

Stylesheet:  
[Folder icon]

☐ Store relative path names (instead of absolute paths)

☒ Always run in foreground

< Back   Next >   Cancel



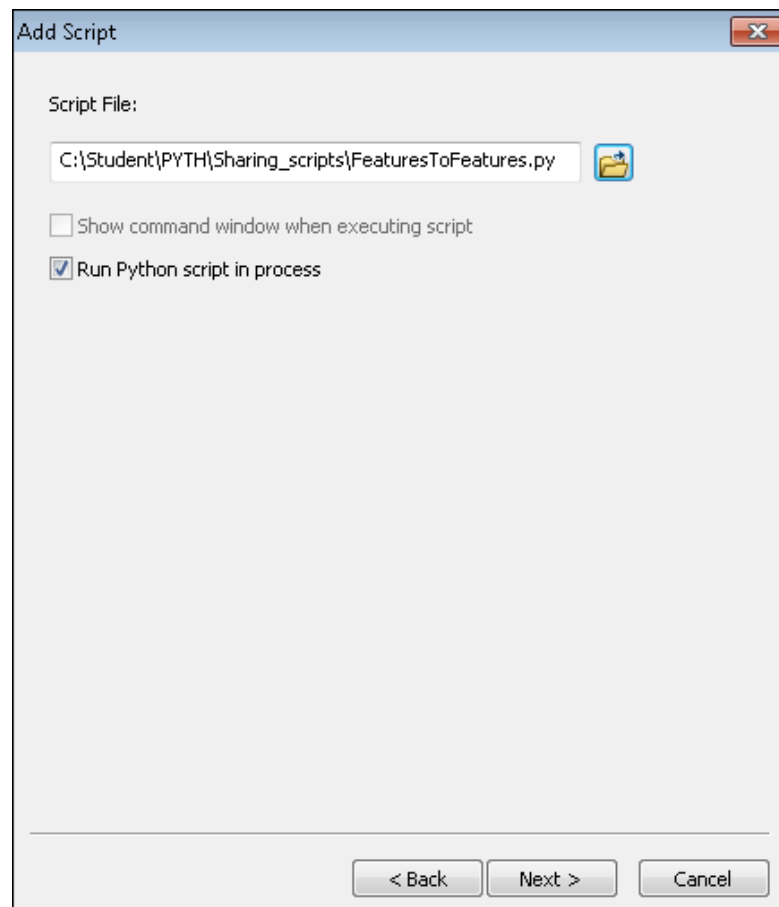
Any time your custom tool is running a script tool, check the Run Python script in process.

**Figure 7.3**

Second panel of the Add Script wizard.

## Second panel

- Point the custom tool to the script that will be executed.
- *Optionally* choose to display the command window (which allows the user to see messages that are written to standard output, such as Python print statements) by selecting the optional check box.



### Third panel

This is where the parameters are set up in the order they are set in the script. Each parameter has additional properties that can be set.

- Whether the parameter is required or optional.
- Whether the parameter value must already exist (input), must not exist (output) or is derived from the script execution, such as a boolean True or False.

Additional parameter properties are available for:

- Applying filters.
- Setting default values.
- Setting environment values that override the current settings.
- Obtaining information from another parameter.
- Applying a layer file to the output data for symbology.

**Figure 7.4**  
Third panel of the Add  
Script wizard.

The screenshot shows the 'Add Script' dialog box with the following content:

| Display Name         | Data Type      |
|----------------------|----------------|
| Input Feature Class  | Feature Class  |
| Output Feature Class | Feature Class  |
| SQL Expression       | SQL Expression |
| Fields to include    | Field Info     |
|                      |                |
|                      |                |
|                      |                |

Click any parameter above to see its properties below.

**Parameter Properties**

| Property      | Value               |
|---------------|---------------------|
| Direction     | Input               |
| MultiValue    | No                  |
| Default       |                     |
| Environment   |                     |
| Filter        | None                |
| Obtained from | Input_Feature_Class |
| Symbology     |                     |

To add a new parameter, type the name into an empty row in the name column, click in the Data Type column to choose a data type, then edit the Parameter Properties.

< Back   **Finish**   Cancel

When running your custom tool as a Server tool, your parameters for input and output data must be set to either *FeatureSet* or *RecordSet*.

## Running a script

Once you have attached your script as a custom tool, the next step is to run your script in ArcGIS Desktop. The script will create a result that can be shared as a package. You have a couple of choices when running your script in ArcMap. You can run it as a *tool dialog* or you can run it in the *Python window*.



Once all parameters are validated, clicking OK executes the script, passing the parameters to the script.

### Running your custom script as a tool dialog

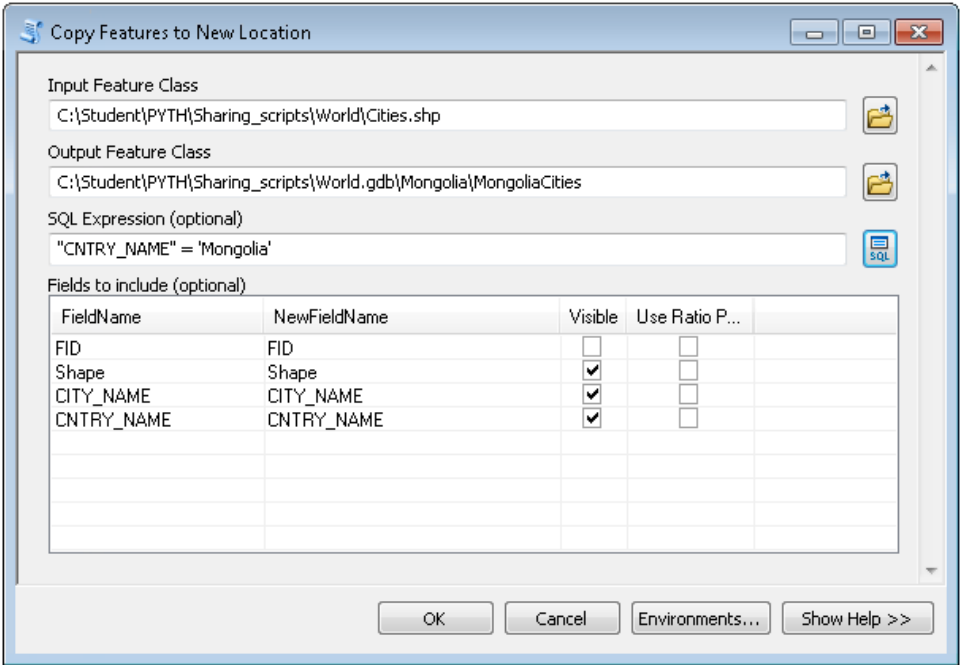
When the script is run as a tool dialog, all of the parameters are filled in the tool dialog entry boxes. Each parameter is validated against the parameter data type.

- For input data, the dialog verifies the data exists.
- For output data, the dialog verifies the data does not exist.

The following graphic is an example of a completed tool dialog.

Figure 7.5

Completed tool dialog.





To rerun geoprocessing tools and scripts for different scenarios: recall the command in the Python window and change the parameter values.

## Running your custom script in the Python window

The Python window is a fully interactive Python interpreter. It provides Python functionality and the ability for geoprocessing tools and scripts to be run directly in ArcGIS.

The Python window is aware of the layers in your map and provides autocompletion functionality for filling in parameter values quicker and easier than tool dialog boxes.

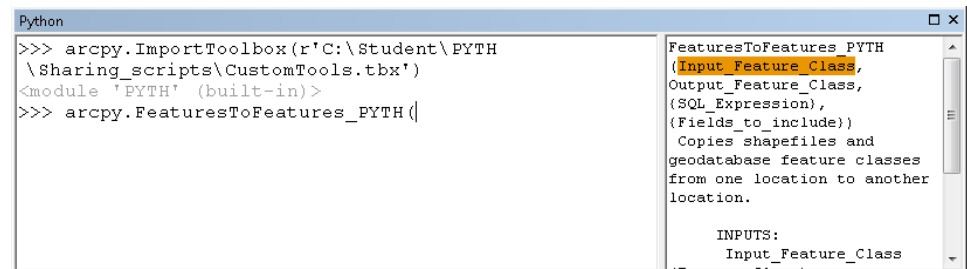
When the custom script is run in the Python window, two steps are required:

1. Importing the custom toolbox using the `arcpy.ImportToolbox()` function
2. Run the script tool using its internal name. This is the Name property for the tool, not the Label property. If there are layers in the data frame that match the data type for the input parameters, they are offered within a drop-down list.

The following graphic provides an example of filling out the parameters for a script tool in the Python window.

**Figure 7.6**

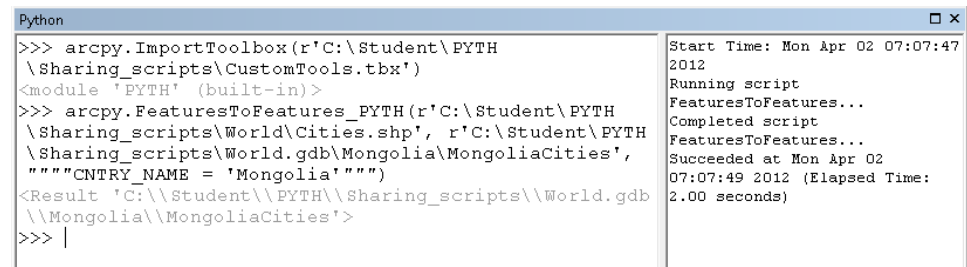
Note that the toolbox alias is part of the tool name, and the Help panel displays the tool syntax.



The following graphic provides an example of what it will look like when the parameters have been filled in and the tool has been executed.

**Figure 7.7**

Note that the result of running the tool is indicated in the Python window; geoprocessor messages are displayed in the Help panel.



The result of running the tool will be stored in the Results window under Current Results. If the script tool creates a new feature class or table as a result, the result is added to the current data frame.

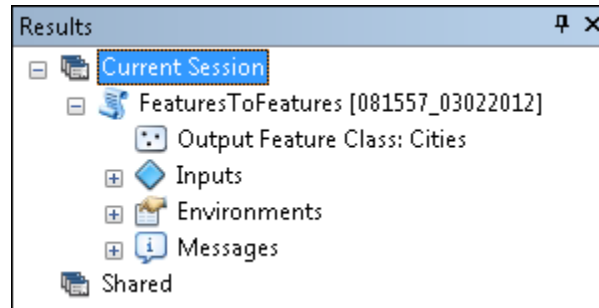
## Sharing a script

Before sharing your results, you must return to your custom script tool and fill out the Item Description. The information you place in the script tool Item Description is stored as part of the package created from sharing the result.

The result to be shared can be accessed from the Results window in the Current Session list.

**Figure 7.8**

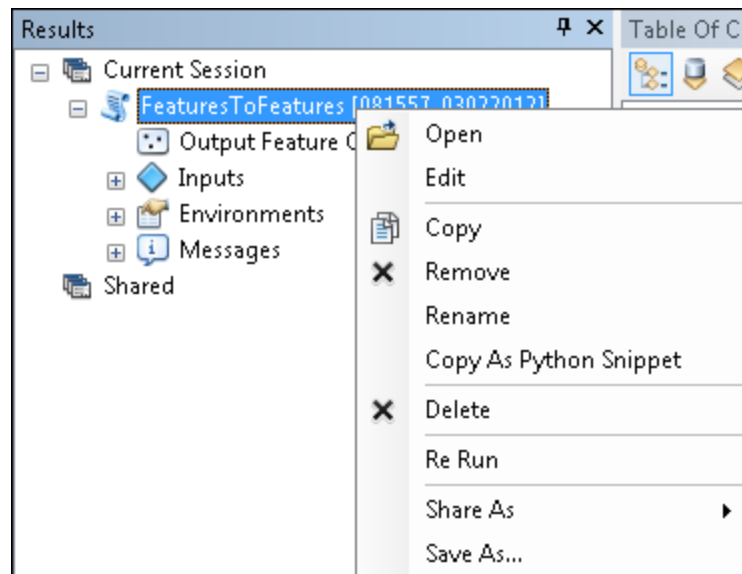
Script tool result in Results window.



Right-clicking the result opens a context menu from which you can rerun the result, open the dialog and run the tool, copy the result as a Python Snippet, and even edit the script.

**Figure 7.9**

Results window context menu.



When the script tool has successfully created valid results, you are ready to share the result as a geoprocessing package or geoprocessing service. You can either store the result as a geoprocessing package to your personal account in ArcGIS Online, or to a local directory.





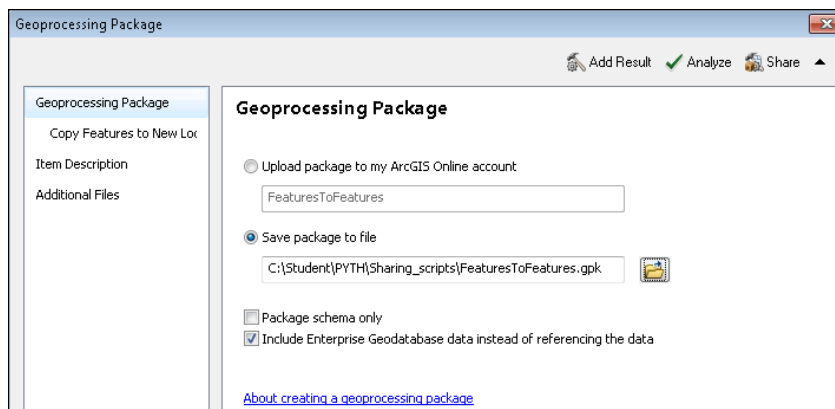
ArcGIS 10.1 Help Library:  
A quick tour of creating a  
geoprocessing package

**Figure 7.10**

Main panel of the  
Geoprocessing Package  
dialog box.

## Creating a geoprocessing package

The main screen of the Geoprocessing Package dialog box is used to determine where the package is to be created and/or uploaded.



The Item Description is populated from the script tool's Item Description.

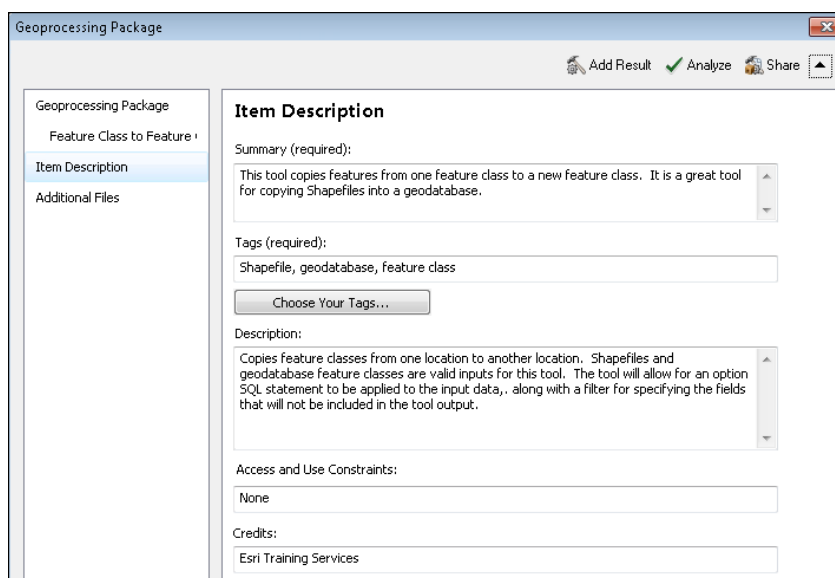


Edit the script tool Item Description values *before* starting the sharing the result process. At a minimum, complete the Summary and add Tags.

The information that you provide in the Item Description helps users determine whether your geoprocessing package is appropriate for their task or workflow.

**Figure 7.11**

Item Description panel of  
the Geoprocessing  
Package dialog box.





To analyze, click the Analyze button on the top of the Geoprocessing package dialog box.

The next step is to analyze the geoprocessing package and resolve any detected errors. When errors are resolved, you are ready to create the package. When you click Share, the result data (both input and output data) and the custom toolbox are packed up into a single file. This file will be placed in the indicated location specified in the main panel of the dialog. After creating a local geoprocessing package, you will receive a confirmation message that the process is complete.

## Unpacking the Geoprocessing Package

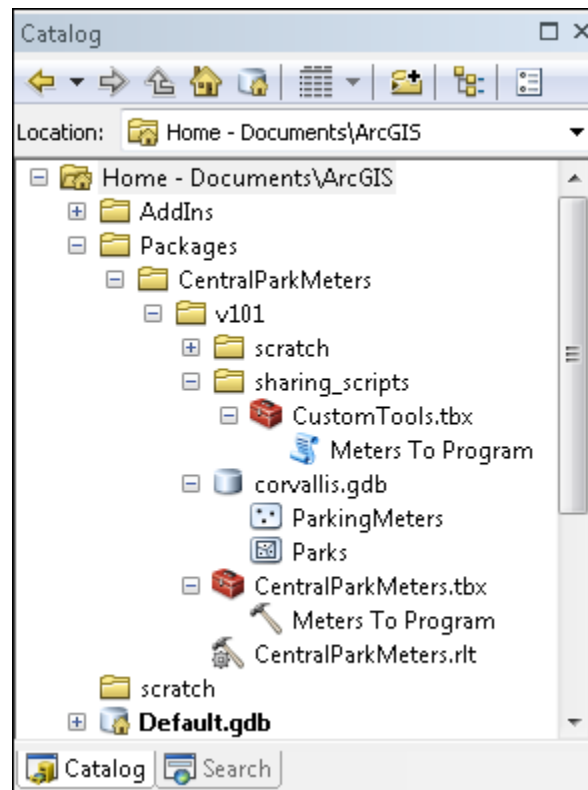
Do one of the following to unpack the geoprocessing package:

- In the Catalog window, unpack the geoprocessing package. (The package is unpacked to your user profile directories.)
- Use the Extract Package geoprocessing tool.
- Drag the package from the Catalog window on to the data frame or Results window.

In the Results window, the Shared list contains the contents of the package. Additionally, in the Table Of Contents window, a group layer is created with the same name as the package.

**Figure 7.12**

An example of the results of unpacking the FeatureToFeatures geoprocessing package.





ArcGIS 10.1 Help Library:  
*A quick tour of authoring  
and sharing  
geoprocessing services*

## Authoring a geoprocessing service

While the course does not explore how to share your result as a geoprocessing service, there are some modifications you may need to make in your script to ensure it runs successfully as a task. The first is to change your parameters for input and output data to either FeatureSet or RecordSet. Any hard-coded paths in your scripts must be fully qualified, and the scratchGDB, scratchFolder, and scratchWorkspace environments must be set. If the service is going to create output that will be used in a map service, then the symbology for the output data should be set to use a layer file.



After filling out the worksheets, you will complete the exercise.

## Plan out a project

You may wish to share your geoprocessing workflow with other users. ArcGIS provides you with the ability to do this.

Successfully sharing your geoprocessing workflows can easily be done with a bit of planning. In this activity, you will step through a scenario and fill out three worksheets. The process of sharing a scripted geoprocessing workflow can be broken down into these high-level steps:

- Determine the development process.
- Determine the geoprocessing workflow.
- Write pseudocode for the script, based upon the geoprocessing workflow.
- Create the script, debug the script, and test the script.
- Attach script to script tool and then run script tool from dialog.
- Share result as a geoprocessing package or geoprocessing service.

### Scenario

The City of Corvallis regularly holds fundraisers in Central Park. A special parking meter rate for each event has been determined for parking meters within a specific distance of Central Park. The city needs to know how many meters to program with the special rate.

The specified distance is 500 feet, Central Park is a feature in the Parks feature class, and the meters to be programmed are features in the ParkingMeters feature class.

Your task is to create a script to automate the process and share it with other GIS users in your organization.

## Determine the development process

This serves as an outline for the script authoring workflow process, along with the subsequent testing and sharing of results to a geoprocessing package.

Planning out the development process helps guide the entire process to a successful completion. *Diagramming the process can help see the bigger picture. The first step might be to write the script in the IDE.*

**Figure 7.13**  
Determining the  
development process  
worksheet.

A spiral-bound notebook is shown from a top-down perspective. The notebook has a silver-colored metal spiral binding on the left side. The pages are white with light blue horizontal ruling. On the page, there are five identical horizontal rectangular boxes, each with a thin black border. Each box contains a single green dot positioned near the left edge. The boxes are arranged vertically with equal spacing between them. The notebook is open to a single page, and the edges of several other pages are visible underneath.

*Notes:*

## Determine the high-level geoprocessing workflow

Things to think about:

- What parameters need to be obtained from the script tool?
- Which layers need to be processed?
- What tools should be used to perform the geoprocessing?
- Are there any required field value updates to the layers?
- How will the results be reported?

Use the following template to indicate the steps in your workflow.

*Depending on your workflow, the first line might be the script parameters.*

**Figure 7.14**

Determine the high-level  
geoprocessing workflow  
worksheet.

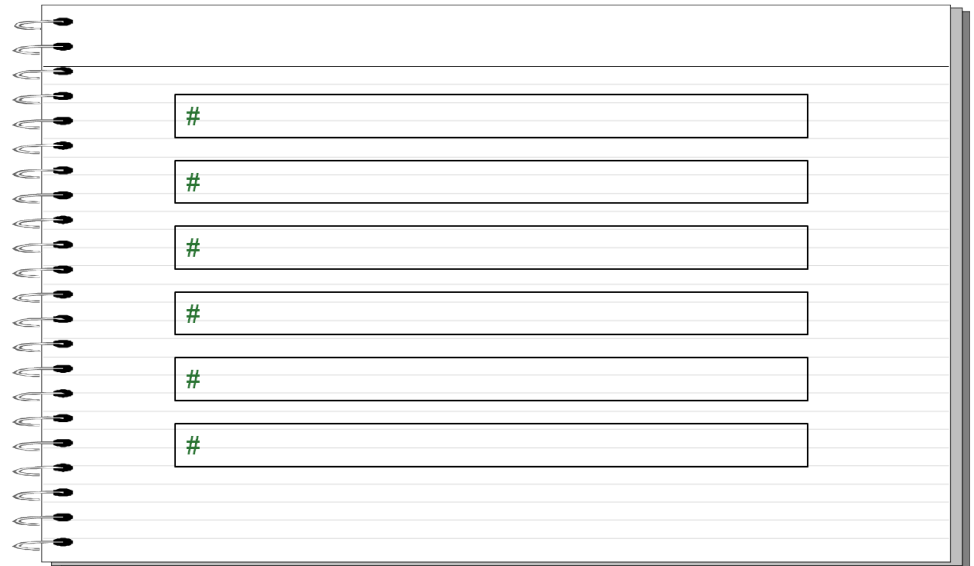
The image shows a spiral-bound notebook with a worksheet template. The template consists of a series of horizontal lines. The first line is a header. Below the header are six rows, each containing a small green dot in the left margin, followed by a rectangular box for notes. The notebook has a spiral binding on the left side.

*Notes:*

## Script the pseudocode

Pseudocode for the script can be added as comments. The comments serve as documentation for the script and also guide the writing of the code. Pseudocode generally follows the same order as the geoprocessing workflow. Fill out the boxes to indicate the pseudocode comments you would add to your script. *Think of these comments as high-level steps in the script.*

**Figure 7.15**  
Script the pseudocode  
worksheet.



A spiral-bound notebook is shown with six horizontal boxes for pseudocode comments. Each box contains a green hash symbol (#) at the beginning, followed by a blank space for text. The boxes are arranged vertically on the right page of the notebook.

*Notes:*

## Write the script

Now that you have planned determined the development process, determined the high-level geoprocessing workflow, and scripted the pseudocode, the next step is to actually create the script and write the code. You will complete this task in the exercise.





70 minutes



## Exercise 7: Share scripts through geoprocessing packages

The City of Corvallis regularly holds fundraisers in Central Park. A special parking meter rate for each event has been determined for parking meters within a specific distance of Central Park. The city needs to know how many meters to program with the special rate. For purposes of this scenario, the specified distance is 500 feet, Central Park is a feature in the Parks feature class, and the meters to be programmed are features in the ParkingMeters feature class.

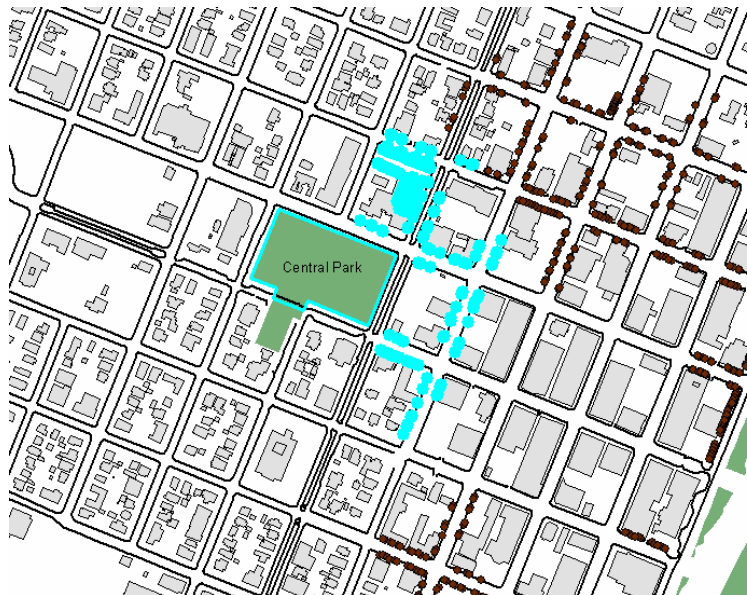
To complete the exercise, choose from one of the following options:

1. Use the provided script.
  - Begin at step 3 and use the MyProject.py document located in the C:\Student\PYTH\Sharing\_scripts folder.
2. Write the script from scratch.
  - Continue with step 1 and complete all steps as directed.

In this exercise, you will:

- Write a script that executes the geoprocessing workflow.
- Attach the script to a custom tool.
- Run the script tool in ArcMap and verify the results.
- Share the results as a geoprocessing package.
- Unpack the geoprocessing package and explore the contents.

**Figure 7.16**  
Result of selecting parking  
meters that are within 500  
feet of Central Park.



## Step 1: Add pseudocode to the script

Once you have created the geoprocessing workflow (in this case, you completed it during the activity), it is good practice to transfer the workflow steps into a format that can be supported in your script. Adding comment lines to your script as pseudocode helps you document your workflow process and serves as a guide for writing the script code. If there is a logic problem in your workflow, it may become apparent as you add your comments.

- a In PythonWin, create a new Python script and save it as **MyCustomProject.py** to the C:\Student\PYTH\Sharing\_scripts folder.
- b Using the geoprocessing workflow, add pseudocode to the script code for each item in the pseudocode.

### Geoprocessing workflow:

- Determine Layers needed for fund raising event:
  - Parks, Parking Meters
    - Need Parking meters within specified distance of Central Park
- Import ArcPy module and set the current workspace
- Set the overwriteOutput environment setting to True
- Obtain script parameter values
  - distance
  - output feature class
- Create Feature Layers
  - Create Parks feature layer for Central Park
  - Create Parking Meters feature layer for selection
- Perform spatial selection
  - Select all Meters that are within specified distance of Central Park
- Copy selected meters to new feature class
- Report selected meter count

- c Save your script and keep PythonWin open.

## Step 2: Write script code

- a In your script, import the required module(s).
- b Set the **arcpy.env** environment values for the data source location.

- c Set the environment value to allow overwriting of output data.
- d Call the appropriate geoprocessing tools to perform the geoprocessing operations.

To assist you in your coding, here are the lessons you can refer to:

| Topic                                      | Lesson   |
|--------------------------------------------|----------|
| Describing data, the Describe object       | Lesson 2 |
| Working with Lists                         | Lesson 3 |
| Working with feature layers and selections | Lesson 4 |
| Accessing and updating field values        | Lesson 5 |
| Working with geometry objects              | Lesson 6 |
| Passing parameter values to scripts        | Lesson 7 |

- e Save your script, then debug and test run the script.



Remember that parameter values are passed to the script in PythonWin through the middle entry box in the Run dialog.

- f Using either the ArcMap or ArcCatalog Desktop applications, verify that correct results were created by the script.
- g If necessary, make any required changes to the script and repeat the testing phase.
- h Once you are satisfied with the script results, comment out the line in the script to overwrite the output data.
- i Close PythonWin.

You have determined the geoprocessing workflow, created pseudocode from the workflow, added the pseudocode to a script as comments for documentation purposes, and written the code from the pseudocode and workflow. Once the script was completed, you debugged the script and ran the script to verify the correct results.

### Step 3: Attach script to custom tool

- a In ArcMap, in the Catalog window, navigate to the C:\Student\PYTH\Sharing\_scripts folder.
- b Right-click the folder and choose New > Toolbox.
- c Provide the name and alias for the toolbox:  
Name: **Custom Tools.tbx**  
Alias: **PYTH**
- d In the Catalog window, right-click the new custom toolbox and select Add > Script.
- e For the first dialog box, provide an internal name for the script tool and an appropriate display name for how it will be displayed in the toolbox.
- f For the second dialog box, browse to the script, and then verify that Run script in process is checked.
- g For the third and final dialog box, fill in the parameters. Make sure they are in the same order as in the script, are using the correct data type for each parameter, and are set for input (must exist) or output (created by script) direction.
- h On the script dialog, click Finish to complete the creation of the custom script tool.



If you are using the script that was completed for you, select MyProject.py. If you wrote your script from scratch, select MyCustomProject.py.



This is a good time to fill out the Item Description for the tool. At a minimum, the Summary, Tags, and Syntax > Dialog Explanation must be filled in before the results can be shared as a geoprocessing package.

- i Right-click your script tool, and choose Item Description.



It may take a few moments to open the Item Description window.

**j** Click Edit and fill out the following sections:

- Tags
- Summary (Abstract)
- Usage
- Syntax: *include explanations for each parameter*
- Credits
- Use Limitation: for example, *Used for Esri Training purposes only*

The Item Description contents are available to anyone accessing your script tool and are placed in the geoprocessing package when the results are shared.

By taking the time to fill out the item description, anyone who comes across your package in ArcGIS Online, or anyone using your script tool, can determine if the package is suitable and if the tool will provide them with the results they need.

**k** Click Save, then close the Item Description Window.

## Step 4: Run script in ArcMap

You have completed the creation of a script and have attached the script to a custom tool. Your next step in the development process is to run the custom tool in ArcMap.

- a** In ArcMap, open the CorvallisMeters.mxd located in the C:\Student\PYTH\Sharing\_scripts folder.
- b** Determine if you want to run the custom tool as a dialog or if you want to run it in the Python window.
- c** If you run the custom tool as a dialog, fill out the required parameters and run the tool.
- d** If you run the custom tool in the Python window, follow these steps:
  - Use **arcpy.ImportToolbox** to import the custom toolbox.
  - Call the custom tool by its internal name along with the appended alias (i.e., `arcpy.ProgramMeters_PYTH()` ).



You can drag the toolbox from the Catalog window into the Python window to fill out the ImportToolbox parameter.

Once the tool completes execution, the result will be added to the table of contents.

- e Verify that the results are correct and that they support the scenario.

If changes need to be made to the script, you can edit the script in PythonWin, save it, and it will update the script in your script tool.

## Step 5: Share results as a geoprocessing package

You just finished running your custom tool in ArcMap and have verified that correct results are being created. In the development process, the final steps are to share the result as a geoprocessing package and verify the geoprocessing package.

When sharing a result as a geoprocessing package, you have a couple of options:

- Share it to your ArcGIS Online personal account.
- Share it as a local geoprocessing package.

In this exercise, you will share the result as a local geoprocessing package.

- a From the ArcMap Geoprocessing menu, choose Results window.
- b From the Results window, expand the Current Session list, right-click the most current result, then select Share As > Geoprocessing Package.
- c In the Geoprocessing Package dialog box, choose the option to Save package to file (accept the default location).
- d On the left panel, click Item Description and verify that the fields are populated.

1. Where did the Item Description data come from?

---

2. In the Geoprocessing Package dialog box, are changes you made in the Item Description contents also reflected in the script tool's Item Description?

---

- e Click Analyze.

A window will display in the ArcMap application. This window contains any errors detected in the result, the script's Item Description, the data, and the toolbox to be packaged.



Most of the time, errors will be generated either for missing Item Description content, missing syntax dialog descriptions, or for data that might have been deleted or moved.

- f** If you need to fix errors, you can right-click the errors, choose an option to fix the errors, then repeat the analysis process.
- g** If there are no errors detected, click Share.

This will package the result data, script, and toolbox into a single .gpk file. When the package is complete, a confirmation pop-up window will display. The confirmation window will show the location of the geoprocessing package file.

- h** Click OK to dismiss the confirmation window.

## Step 6: Verify the geoprocessing package

Finally, you will explore the contents of the geoprocessing package.

- a** In the Catalog window, navigate to the C:\Student\PYTH\Sharing\_scripts folder and locate your local geoprocessing package file (.gpk).
- b** Right-click the local geoprocessing package file, then choose Unpack.

The packaged result will be placed in the Shared list in the Results window.

- c** In the Catalog window, navigate to the Home - Documents\ArcGIS node and if necessary, expand the node.



If the Home - Documents\ArcGIS node is not displayed in the Catalog window, you can access the folder by creating a Connection to C:\Users\Student\Documents.

- d** Expand the Packages folder. Locate your package name and then explore the contents of the v101 folder.
- e** Use the Catalog window to help answer the following questions.

3. Under which folder is the input data located?

---

4. In which folder is the result file (.rlt) located?

---

5. Does the toolbox located in the v101 folder contain your custom script tool?

---



## Lesson review

1. How can you make a Python script dynamic?

---

---

2. What are the steps needed to share a script as a geoprocessing package?

---

---

---

3. List the basic workflow for sharing GIS workflows with others.

---

---

---

## Answers to Lesson 7 questions

### Exercise 7: Share scripts through geoprocessing packages (page 7-21)

1. Where did the Item Description data come from?

**The fields were populated from the script tool's Item Description.**

2. In the Geoprocessing Package dialog box, are changes you made in the Item Description contents also reflected in the script tool's Item Description?

**No, changes made in the Geoprocessing Package dialog are not saved to the tool.**

3. Under which folder is the input data located?

**The input data is located under the v101 folder.**

4. In which folder is the result file (.rlt) located?

**The result file is located in the v101 folder.**

5. Does the toolbox located in the v101 folder contain your custom script tool?

**Yes, it is a copy of the script tool, contained in the custom toolbox, located in the Sharing\_scripts folder.**

# 8

## Debugging scripts and handling runtime errors

### Key terms

errors

exceptions

try-except

### Introduction

You have used several techniques to find syntax errors in your scripts. Visually checking the code is one way to find errors, and using the Check button in the PythonWin main toolbar is another way to identify errors.

In this lesson, you will explore workflows for debugging your scripts and handling any errors generated when the script is executed.

### Topics covered

- Debugging scripts with IDE tools
- Visually finding errors
- Debugging workflows
- Catching and handling runtime exceptions

### Learning objectives

After completing this lesson, you will be able to:

- Debug scripts in PythonWin.
- Trap and handle runtime exceptions.
- Use the Python traceback module.

## Debugging workflow

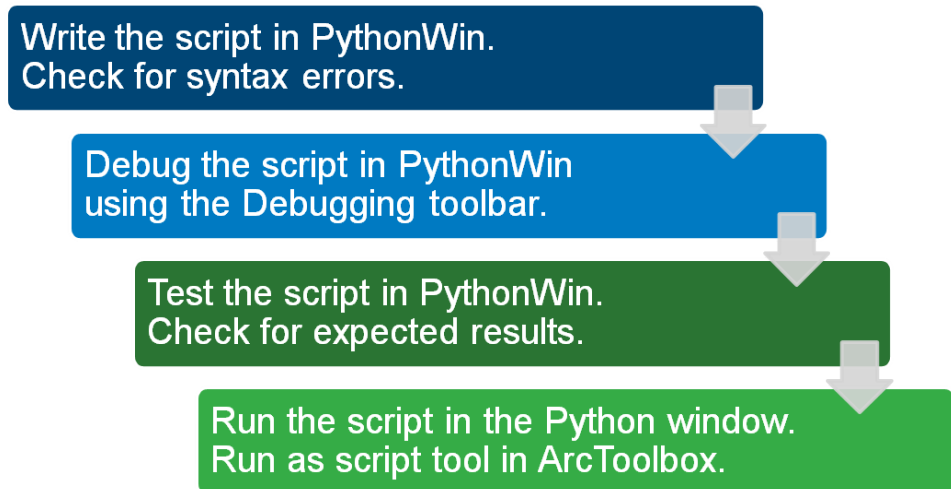
*What are some benefits to debugging your scripts and handling any errors that might occur?*

When debugging your script, the PythonWin and IDLE applications provide you with the ability to check for syntax errors, comment your code, and print values to a window. These applications also provide functionality for stepping through your code line by line and examining variables.

You have used PythonWin throughout the course to develop code, check for syntax errors, and run code. In this lesson, you will use PythonWin to debug your scripts and write code to trap errors.

### A typical workflow for debugging scripts:

**Figure 8.1**  
Typical workflow for  
debugging scripts.





ArcGIS 10.1 Help Library:  
Executing and debugging  
Python



To access the Debug Control window: from the Shell menu, select Debug > Debugger. Step through the script using the Out button.



In the Run Script dialog box, for the debugging option, choose Step-through, then click OK.

## Debugging scripts in PythonWin

When you debug your script, make sure you do it in the right place. Both the PythonWin application and the Python window in ArcMap are excellent locations to debug and test your code.

PythonWin offers numerous resources for successfully debugging your code, including a tool for checking code syntax, the Interactive Window for print and command functionality, and the Debugger toolbar.

### Checking for syntax errors

Writing your script in a Python IDE, such as PythonWin or IDLE provides you with the functionality to easily check for and correct any syntax or indentation errors.

In *IDLE*, you can use the Debug Control window on a script. In the Debug Control window, you can step through the script and view the variable values for each step. Any exceptions generated will be displayed in the Debug Control window.

In *PythonWin*, if a syntax error is detected upon using the Check button on the Standard toolbar, the cursor is placed at the location where the error was detected. You can then correct the syntax and check the script again until there are no errors.

### Debugging scripts in PythonWin

The *Watch window* allows you to view the values of any expressions you define. The *Stack window* allows you to view the values for the variables at that step. The cursor is placed on the first line of code being interpreted by Python.

### Stepping through code

Click the Step Over button to prompt the Python interpreter to execute the line of code and then move to the next line of code. Repeat this action until you need to view variable values or you reach a line of code that you would like to examine further.

An alternative to stepping through multiple lines of code is to set a breakpoint at a line of code. Place your cursor on the line of code where the breakpoint is to be set, and then click the Toggle Breakpoint button. Click the Go button to execute the script up to the breakpoint.

**Examining variables**

The Stack window is used to examine the values of variables in your script. To navigate to the variables, expand the <module> node and then expand the Locals (Dict) node. Each variable will be an item in the node and starts with `VAL=`.

If you are looping through a list of items in a Python List, the Watch window can be very useful. In the Watch window, click <New Item> and type in the name of the loop variable. Press Enter and you will see the value of the variable. When stepping through the loop, each iteration will display the value of the variable for that iteration.

**Commenting code**

You typically use comments in your code to document the script workflow. Commenting code can be very useful when debugging your script. When the script is run, any error messages received may indicate an approximate location where the error was detected.

Commenting blocks of code and then running the script can help narrow down the location of the error. For example, your script performs a Buffer operation on data, then a Clip operation and then finally an Erase operation. You notice that the Erase operation generated the error. By commenting out the code from the Erase operation down to the end of the script, you can then run the script again. If the script completes successfully, now you have a starting point at the Clip tool for further debugging of your script.

**Using print statements**

Adding print statements can also help you debug your script. Print statements can be used to indicate steps in the script, and for printing variable values throughout the script. Taking the previous example, if you added a print statement for the Buffer tool, a print statement for the Clip tool and a print statement for the Erase tool, then it would be easy to see how far the script progressed before the error was generated. This would give you a good starting point for further debugging.

**Verifying results**

As a final step of debugging, you should verify that the script created the correct results. The ArcMap and ArcCatalog applications can be used to verify that the data was created and/or the correct results were generated.

## Handling script exceptions

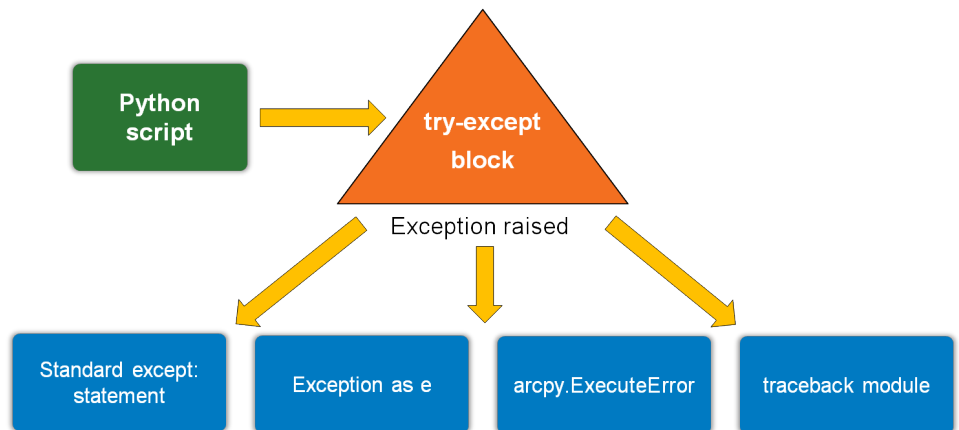
When writing scripts, it is almost impossible to avoid errors. Fixing syntax errors is fairly straightforward, but finding errors that occur when the script executes can be more challenging.

When a tool returns an error message, ArcPy raises a system **error**, or **exception**. An error can be detected and must be corrected before the script is run. An exception is an error that is detected during the execution of the script and is returned from Python or the Geoprocessor. Python provides the ability for you to write a routine that can run when the system error occurs. By handling the system error, you can manage the errors and improve the usability of your script instead of having the script crash.

### Using the try-except block

Python provides the **try-except** block to wrap portions of code or even the entire script. Any error detected in the try statement will be handled by an except statement. There are several types of exception classes available. They are detailed in the following graphic.

**Figure 8.2**  
Handling exceptions in  
Python.



The code that you want to run in your script is placed in the *try* statement. If an error occurs, an exception is raised, and the code in the except block is then executed.

Code that you place in the except block could include print statements for detailing the exception or fetching the geoprocessing messages, and then reacting accordingly.

## Working with exceptions

Table 8.1

| Techniques used to handle exceptions                    |                                                                     |
|---------------------------------------------------------|---------------------------------------------------------------------|
| Exception class                                         | Technique                                                           |
| standard <b>except</b> statement                        | Catches any error.                                                  |
| <b>Exception as e</b> block                             | Catches any error. <b>e</b> contains the message.                   |
| <b>arcpy.ExecuteError</b>                               | Catches only arcpy geoprocessing errors.                            |
| <b>traceback</b> module in standard <b>except</b> block | Provides detailed Python messages and shows geoprocessing messages. |

### Using Exception as e

Using the `except Exception as e` statement can be useful when you need error messages on the exception that occurred. The `except Exception as e` statement will catch both Python and Geoprocessing exceptions.

The error message for the exception that occurred is stored in **e**. You can print the error messages using `print e` in the `except` statement.

Example code using a simple `except Exception as e:` statement:

```
try:
    #
    #
    # Your code goes here
except Exception as e:
    # print the error messages
    print "An error has occurred"
    print e
```



## Using `arcpy.ExecuteError`

There may be times when you want to handle an exception raised by the Geoprocessor separately from Python exceptions. The `arcpy.ExecuteError()` and `arcpy.ExecuteWarning()` exception classes are raised when a geoprocessing tool encounters an error or warning. You can use the exception class in an `except` statement along with a `print` statement to display the geoprocessing error message or warning message.

Example code using an `arcpy.ExecuteError` exception class:

```
try:
    #
    #
    # Your code goes here
except arcpy.ExecuteError:
    # print the geoprocessing error messages
    print "A geoprocessing error has occurred"
    print arcpy.GetMessages(2)
except Exception as e:
    # print the Python message
    print "An error has occurred"
    print e
```

If you are catching both geoprocessing exceptions and Python exceptions in the same `try-except` statement, place the `except arcpy.ExecuteError` statement above a standard `except` statement or `except Exception as e` statement to guarantee that the Geoprocessor exception is handled separately from the Python exception.



## Using the traceback module

The `traceback` module can be useful in debugging your scripts, tracking down why the script failed, and identifying the line of code where the error occurred. To obtain this information, the `traceback` module is used in conjunction with the `sys` module to get information on the system error and the location of the error.

When implementing the `traceback` module, place it in a standard `except` statement. When the system error occurs in the `try` statement, the `except` statement will get the `traceback` object and print out the process information along with the error messages.

If the script is being run as a script tool, the `arcpy.AddError()` function can be used to pass the `traceback` and geoprocessing messages to the Progress dialog box, the Results window, and the Python window.

Code example using the `traceback` module:

```
import arcpy
import sys
import traceback
try:
    #
    # Place code here to execute
except:
    # Error occurred
    tb = sys.exc_info()[2]
    tbinfo = traceback.format_tb(tb)[0]
    pymsg = "PYTHON ERRORS:\nTraceback Info:\n" + \
        tbinfo + "\nError Info:\n" + \
        str(sys.exc_type) + ": " + \
        str(sys.exc_value) + "\n"
    arcpy.AddError(pymsg)

    msgs = "GP ERRORS:\n" + arcpy.GetMessages(2) + "\n"
    arcpy.AddError(msgs)
    print pymsg + "\n"
    print msgs
```

## Getting geoprocessing messages

In your script, you can also incorporate geoprocessing messages to provide more information about the messages the script generates. The `arcpy.GetMessages()` function returns messages from the geoprocessor based on their severity level.

### Syntax:

```
arcpy.GetMessages({severity})
```

Table 8.2

Three severity levels

| severity | Messages returned    |
|----------|----------------------|
| 0        | Informative messages |
| 1        | Warning messages     |
| 2        | Error messages       |

Example code using a simple except statement:

```
try:
    #
    #
    # Your code goes here
except:
    # print the error messages
    print "An error has occurred"
    print arcpy.GetMessages(2)
```

Adding custom geoprocessing messages

The ArcPy module provides three functions for creating custom geoprocessor tool messages that can be accessed by any of the GetMessage functions.

ArcPy functions for adding messages to geoprocessor

| Function           | Severity | Message type         |
|--------------------|----------|----------------------|
| arcpy.AddMessage() | 0        | Informative messages |
| arcpy.AddWarning() | 1        | Warning messages     |
| arcpy.AddError()   | 2        | Error messages       |

Table 8.3

These messages also display in the progress dialog box, the tool's result, and the Python window.



Upon completion of the script, ArcPy will raise a system error and the tool or model will stop execution.

When you add an error message using `arcpy.AddError()`, the script will continue to execute. At that point, you need to add code to handle the error condition, such as closing cursors and removing intermediate data.

Example code using `arcpy.AddError` and `arcpy.AddMessage`:

```
import arcpy
import os
try:
    # Variables
    fc = arcpy.GetParameterAsText(0)
    newFC = arcpy.GetParameterAsText(1)
    SQLExp = arcpy.GetParameterAsText(2)
    fLyr = os.path.basename(fc) + "_lyr"

    # Create feature layer, applying SQL expression
    arcpy.MakeFeatureLayer_management(
        fc, fLyr, SQLExp)

    # Obtain result from GetCount tool
    result = arcpy.GetCount_management(fLyr)
    featCount = int(result.getOutput(0))

    # Copy features to new feature class
    if featCount == 0:
        arcpy.AddError(fLyr + " has no features")
        raise arcpy.ExecuteError
    else:
        arcpy.AddMessage("Copying features")
        arcpy.CopyFeatures(fLyr, newFC)
except Exception as e:
    print "An error occurred"
    print e
```

## Tips and best practices

- Use exception handling techniques in your scripts to remove script crashes, reduce user frustration, and implement custom handling of system errors.
- Incorporate `try-except` blocks in your scripts (either for portions of the script or the entire script).
- Use the `traceback` module as a tool only for debugging your scripts. Comment the `traceback` code before deploying the script.
- Filter geoprocessor messages returned by `arcpy.GetMessages()` by passing the severity level to the function.
- Messages can be obtained from a tool using a result object. Many of the geoprocessing tools will return a result object when the tool successfully executes. Messages are returned using the syntax `result.getMessages(<n>)`, where `<n>` is the severity level of the messages.



55 minutes



## Exercise 8: Debug scripts and handle exceptions

Scripts can be more effectively written and deployed by using debugging techniques and including code to handle any errors that may occur when the script is executed. You will explore how to detect, handle, and report the Python and geoprocessing errors that may occur.

In this exercise, you will:

- Locate and fix Python syntax errors.
- Incorporate a try-except block in an existing script.
- Implement Python exception classes and the `arcpy.ExecuteError` exception class in your script.
- Report on geoprocessing error messages and add custom messages.
- Report the Python or geoprocessor error messages and indicate the location of the error in the script.

## Step 1: Debug the script

The exception handlers that you will work with in the exercise use a `try-except` statement. The `try-except` statement is a simple Python structure, where you place the code to run in the `try` statement and the code to handle the `exception` in the `except` statement.

- a In PythonWin, open the `WorkingWithExceptions.py` script, located in the `C:\Student\PYTH\Debugging_scripts` folder.
- b Save the script as **`MyWorkingWithExceptions.py`**.
- c Read the script comments so that you understand the script workflow.
- d From the File menu, select Run.

The script did not run, due to a syntax error. PythonWin displays a syntax error message. The cursor is placed at the location where the syntax error was detected.

- e Fix the error, then check for and fix any additional syntax errors in your script, but do not run your script yet.

You will run the script in the next step.

## Step 2: Incorporate a try-except block

In this step, you will incorporate the `try-except` blocks into your script and explore working with a simple `except` statement.

The `try-except` block is a simple Python structure, in which the code you wish to run is placed in the `try:` statement, and the code to handle the exception is placed in the `except:` statement.

In the previous step, the script contained syntax errors that prevented the script from running. Now that the syntax errors have been fixed, you are ready to run the script.

- a Run the script and view the output in the Interactive Window.

The script failed with an `ExecuteError` exception and printed a lot of error messages.



A good script always has code to handle exceptions and print more meaningful messages, which provide a more positive experience to the end user.

Your first task is to add a `try-except` block to the script. The `except` statement will catch the exception and print any geoprocessing messages.

- b** In the script, locate the comment *# Variable assignments for GP tasks* and move to the line above the comment.

- c** Incorporate a `try-except` block into the script. The `try:` statement should include the comment line and all lines of code below the comment.

*Hint:* Select all lines of code below the **`try:`** statement. Press the <Tab> key to indent the selected code.

- d** Navigate to the bottom of the script and press the <Enter> key twice to dedent the cursor to the left side of the script window. Add an **`except:`** statement to the script.

- e** Inside the `except:` statement, add the following lines of code:

```
print "An error occurred"
print arcpy.GetMessages()
```



Be sure the code is indented so that it will execute when Python jumps to the `except:` statement.

- f** Check the script syntax, close PythonWin, then reopen the script.

*Why close and then reopen PythonWin?*

The `arcpy.GetMessages()` function will return all geoprocessor messages that are generated in the current session. These messages need to be cleared from memory, so you will close PythonWin and then reopen the script in PythonWin.

- g** Run the script.

1. Did the script fail to execute?

---

2. What geoprocessing tool error has occurred?

---

---

3. Based on the error messages, how might you fix the problem?

---

---

It appears that the error is occurring in the `SelectLayerByLocation` tool, but you can't be sure of the cause. It might be a Python error or an incorrect parameter passed to the tool.

**h** In the next step, you will work with the `Exception as e` statement.

### Step 3: Incorporate an Exception as e statement

You can use the `except Exception as e` statement to print error messages to the Interactive Window in PythonWin. In addition to any Python exceptions, any exceptions raised by the geoprocessor will also be handled. The error messages can be printed to the Interactive Window or added to the Geoprocessing progress dialog box, the Python window, and the Results window.

**a** In your script, comment out the entire `except:` statement.

In the previous step, a geoprocessing error was displayed by the `except:` statement and indicated the error was with the `arcpy.SelectLayerByLocation` function. You will narrow down the cause of the error by examining the output feature layer created by the tool.

**b** In the Interactive Window, press the Enter key to obtain a prompt.

**c** In the Interactive Window, use the `arcpy.GetCount_management` function to return the number of features in the **fc2Lyr** feature layer.

4. How many features are in the feature layer?

---

Diagnosis:

- There are no features in the `RailLyr` feature layer.
- The error message indicates an expected field was not found or retrieved properly when a selection was performed on the feature layer.

**d** Examine your script and locate any code that specifies fields for `fc2Lyr`.

The `SQLExp` string is used by the `arcpy.MakeFeatureLayer_management` function to create `fc2Lyr`.

**e** Start ArcCatalog 10.1 and navigate to `C:\Student\PYTH\Debugging_scripts\SanDiego.gdb`.

**f** Open the properties for the Railroads feature class and view the Fields tab in the Feature Class properties.

5. Is the field specified in the `SQLExp` string spelled correctly?

---

**g** Close ArcCatalog and return to PythonWin, then make the required change in the script.

**h** Navigate to the bottom of the script and add the **`except Exception as e:`** statement.

**i** Inside the `except Exception as e:` statement, add the following code:

```
print "Exception as e handler"
print e
```

**j** Check the script syntax.

**k** Close PythonWin, reopen the script in PythonWin, then run the script.

6. Did an error occur?

---

Diagnosis:

- The print statement executed just before the code that creates the SearchCursor.
- It appears that there might be a problem with the parameter containing the list of fields passed to the cursor. Each field name specified in the field list is a column in the feature layer's attribute table.

- l** Leave PythonWin open and start ArcCatalog.
- m** In ArcCatalog, navigate to C:\Student\PYTH\Debugging\_scripts\SanDiego.gdb.
- n** Right-click MajorAttractions and choose Properties.
- o** Click the Fields tab and verify that each field specified in the fields variable is correct.

7. Which field name is incorrectly specified in the fields variable?

---

- p** In the script, make the required change to the field name, then check the script syntax.
- q** Close PythonWin, reopen the script in PythonWin, then run the script.

8. Did an error occur?

---

Diagnosis:

- The field names parameter is not the problem.
- The problem might be with the cursor or with the `for-in` loop code.

Your next task is to see if you can narrow down the error further. The error could be a Python error or a geoprocessor error.

- r** In the next step, you will implement the `arcpy.ExecuteError` exception class in your script.

## Step 4: Incorporate an `arcpy.ExecuteError` exception class

- a** The `arcpy.ExecuteError` exception class will catch only geoprocessor exceptions that are at the error severity level. If the error is a geoprocessing error, it will be caught by this handler.

- b** In your script, comment the `except Exception as e:` statement and code.
- c** Add the `except arcpy.ExecuteError:` statement to the bottom of your script.
- d** Add the following lines of code to report the geoprocessor error messages:
 

```
print "A geoprocessing error has occurred."
print arcpy.GetMessages(2)
```
- e** Add a final line to your script to indicate that the script has completed.
- f** Check for syntax errors, close PythonWin, then reopen the script in PythonWin.
- g** Run the script.

9. Did an error occur?

---

To obtain more detailed information about the error, you will use the `traceback` and `sys` modules.

## Step 5: Use the `traceback` module

The `traceback` module can be useful in debugging your scripts to track down why and where the script failed, and the line of code where the error occurred. To obtain this information, the `traceback` module is used in conjunction with the `sys` module to gather information on the system error and the location of the error.

- a** In your script, comment the `except arcpy.ExecuteError:` statement and code.
- b** Navigate to the bottom of the script and add a standard `except:` statement just above the final print statement.

The ArcGIS 10.1 Help Library contains many Python code examples that you can use in your scripts. By copying the example to the clipboard and then pasting it into your script, you can easily implement the example's functionality.

- c** In the ArcGIS 10.1 Help Library, search for **traceback**.
- d** In the search results, click Unzip Python script.

- e In the Help topic, scroll down and locate the `except:` statement. Copy the code in the statement.
- f Return to PythonWin and paste the contents of the clipboard into the `except:` statement.
- g Add the following lines of code to the `except:` statement to print the error messages:

```
print pymsg + "\n"
print msgsg
```
- h Scroll to the top of the PythonWin script window and locate the `import arcpy` statement.
- i Add a new line under the import statement and write code to **import** the **sys** module. Add a second line of code to **import** the **traceback** module.
- j Check the script for syntax errors, close PythonWin, then reopen the script.
- k Run the script.
- l In the Python Interactive window, scroll through the messages.

The messages clearly state that the error is a Python error, and the line where the error occurred is indicated.

- m Return to the script window, fix the code error, then check the script for syntax errors.
- n Close PythonWin, reopen the script in PythonWin, then run the script.
- o In the Python Interactive window, scroll through the messages. The messages state that the error is a GP error, and the line where the error occurred is indicated. In the Interactive Window, continue to scroll up until you locate the PYTHON ERRORS: messages.
- p The PYTHON ERRORS messages show the line of code where the error occurred.

## Summary:

- The error (Error 000725) is a geoprocessing error.
  - The output dataset specified for the **arcpy.CopyFeatures\_management** function already exists.
  - Based on this information, you could add code in your script to check for the existence of the output feature class before running the **arcpy.CopyFeatures\_management** function, or you could provide a warning message to the end user that the feature class already exists.
- q If you have the time and would like to perform one additional step, proceed to the provided Challenge Step. Otherwise, close PythonWin.

## Challenge: Challenge



5 minutes

- a Modify the script so that the `arcpy.ExecuteError` exception class catches the GP exceptions and the standard `except:` statement catches the Python exceptions.



Remember to restart PythonWin to clear the geoprocessor messages.



## Lesson review

1. Of the error handling techniques discussed in the class, which technique returns the most amount of detail?

---

---

2. Which error handlers should you use if *only* geoprocessing errors are to be handled?

---

---

---

3. Write an except block that adds a custom error message to the Progress dialog box, the Python window, the Results, and the Interactive Window in PythonWin.

---

---

---

---

---

4. What functionality is available in PythonWin for stepping through your code?

---

---

## Answers to Lesson 8 questions

### Exercise 8: Debug scripts and handle exceptions (page 8-13)

1. Did the script fail to execute?

**Yes.**

2. What geoprocessing tool error has occurred?

**ERROR 999999: Error executing function.**

**An expected Field was not found or could not be retrieved properly.**

3. Based on the error messages, how might you fix the problem?

- **Verify the feature layers were created correctly.**
- **Check for an error in the `arcpy.SelectLayerByLocation_management` tool parameters.**

4. How many features are in the feature layer?

**0**

5. Is the field specified in the SQLExp string spelled correctly?

**No, it should be spelled "STREET\_NAM".**

6. Did an error occur?

**Yes. A column was specified that does not exist.**

7. Which field name is incorrectly specified in the fields variable?

**ADR. Instead, it should be ADDR.**

8. Did an error occur?

**Yes, name 'wow' is not defined**

9. Did an error occur?

**Yes. The script crashed with a NameError exception: name 'wow' is not defined.**

## Challenge solution: Challenge

- Uncomment the `except arcpy.ExecuteError:` statement and code.
- Place the `except arcpy.ExecuteError:` statement above the standard `except:` statement.



# 9

## Automating map production

### Key terms

layer file

layout element

map document

map layer

### Introduction

Many organizations maintain a large number of map documents that are used for a variety of reasons. The `arcpy.mapping` module provides the ability to automate updating of map document components (such as layers and layout elements) without the need to open the every single map document in ArcMap. This enables the scripting of map document updates to be automated.

### Topics covered

- Working with a map document
- Listing map document contents
- Managing layers
- Publishing maps
- Exporting maps and reports

### Learning objectives

- Automate the updating of map documents in a script.
- Update map layer properties in a map document.
- Update map layout properties in a map document.

## The arcpy.mapping module

The arcpy.mapping module contains functionality for working with existing map documents (.mxds) and for printing and exporting map document contents. The arcpy.mapping module is intended not as a replacement for ArcObjects code, but to provide easy-to-use functions that take the place of many lines of ArcObjects code. This makes the arcpy.mapping module perfect for scripting tasks when manipulating mxd content and for output to graphic formats.

Tasks that can be accomplished using the arcpy.mapping module:

- Inventory the contents of map documents, such as layer properties, data sources, data frame properties, layout elements, and broken data sources.
- Update layer data sources or repair broken layer data sources.
- Update layer symbology.
- Update or change layout elements.
- Save a map document to a prior ArcGIS version (useful for sharing .mxd files).
- Update map document metadata, such as title, author, keywords, summary.
- Export map contents to a graphic format.
- Using Data Driven Pages; create a map book series in a PDF (such as a reference map book or thematic map book).

***When working with a map document, what types of changes do you typically make?***

## Terms commonly used when working with the arcpy.mapping module

Table 9.1

| Term                     | Definition                                                                                                                                                    |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>layer file</b>        | References geographic data and is stored as a file on disk. Contains the same information as a layer in the ArcMap's Table Of Contents window.                |
| <b>layout element</b>    | A container for an item placed on the layout, such as graphics, legends, dataframes, north arrows, scalebars, pictures, and text.                             |
| <b>map document</b>      | A container for objects in a map, such as data frames, layers, tableviews, page layouts, reports, graphs.                                                     |
| <b>map layer</b>         | Defines how datasets are symbolized and labeled. In a map document, a layer stores the map layer properties.                                                  |
| <b>data driven pages</b> | Provides the ability to generate a series of map pages from a single layout, using a index layer in the dataframe. Properties are stored in the map document. |



## Using the arcpy.mapping module to modify map document contents

The arcpy.mapping module is included as part of the ArcPy site package install. Its main purpose is to provide the ability to work with and manipulate the contents of existing map documents (.mxd) and layer files (.lyr).

**arcpy.mapping module map document functionality**

| Function                   | Description                                                                                                                                                                                                                                        |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Reference the map document | The .mxd file can be referenced on disk by providing a path to the file or by referencing a map document loaded into the ArcMap application with the "CURRENT" keyword.                                                                            |
| List contents              | List functions include: Bookmarks, BrokenDataSources, DataFrames, Layers, LayoutElements, and TableViews.                                                                                                                                          |
| Manage layers              | AddLayer, AddLayerToGroup, InsertLayer, MoveLayer, UpdateLayer, and RemoveLayer are included for managing layers. The Layer function is used to reference a layer file.                                                                            |
| Manage layout elements     | The ListLayoutElements function can be used with a filter to return only the elements you are interested in. Layout element types returned include: DataFrame, GraphicElement, LegendElement, MapsurroundElement, PictureElement, and TextElement. |
| Publish maps               | Functions are included to create the Service Definition Draft files, analyze the map prior to the creation of the SD file, stage the SD file, and then upload the SD to the server.                                                                |
| Export maps and reports    | Map documents can be exported to all supported graphic formats. A report authored in ArcGIS Desktop can be exported to HTML, PDF, TIFF, TXT, and XLS formats.                                                                                      |

Table 9.2



## Referencing the map document

Prior to working with the functionality provided by the `arcpy.mapping` module, a reference to the map document must first be obtained. Depending on where you are accessing the map document, the procedure will be slightly different. In both cases, a `MapDocument` object will be returned.

### Provide a path to the mxd file on disk.

This is the recommended method to access the map document. This method allows Python to access the map document outside of an ArcGIS application. A script written using this method can also be run inside the ArcMap or ArcCatalog applications.

### Access the map document through the Python window

This is done using the "CURRENT" keyword instead of a path to the map document. This allows Python to access the map document that is loaded into the ArcMap application. This method allows for testing code syntax and functionality that will be later written into a script.

Script tools that use the "CURRENT" keyword must be run from ArcMap (either as a dialog or loaded into the Python window).

When using the "CURRENT" keyword in a script, be sure to disable the background geoprocessing option in ArcMap. When background processing is enabled, any script run will execute as if they were being run as a standalone script outside of the ArcGIS application. As a result, there is no "CURRENT" map document loaded into memory. When creating a script tool that uses the `arcpy.mapping` module, be sure to confirm that the option to always run in foreground is enabled in the script tool.

Code examples for obtaining a map document reference:

```
# Reference the map document on disk
mxDoc = arcpy.mapping.MapDocument(
    r"C:\Student\PYTH\CityBase.mxd")
print mxDoc.title

# Reference the map document in memory
mxDoc = arcpy.mapping.MapDocument("CURRENT")
print mxDoc.filePath
```



If the script tool is run in ArcCatalog, there is no map document loaded into the application and a runtime exception will be thrown.

## Working with a map document template

There is no functionality in the `arcpy.mapping` module for authoring a brand new map document from scratch. However, a map document can be created from a map document template. Once changes are made, a copy of the map document can be saved.

An example of this is working with an existing map document template containing pre-existing data frames, layout elements (with page size and orientation already set up and default metadata assigned), and then using the `arcpy.mapping` functionality to manipulate the contents of the map document template. When finished, the `saveAs()` method needs to be used on the map document object in order to create the working map document file, and then delete the `MapDocument` object from memory.

## Listing map document contents

The `arcpy.mapping` module provides a number of list functions.

### `arcpy.mapping` functions for listing map document contents

Table 9.3

| Function            | Description                                                                                                                                                                                                                           |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ListDataFrames*     | A Python List of dataframe objects is returned. Dataframe object properties include credits, description, extent, mapUnits, spatialReference. Methods include panToExtent, zoomToSelectedFeatures.                                    |
| ListLayers*         | A Python List of Layer objects is returned. Layer object properties include dataSource, description, isFeatureLayer, name, symbologyType. Methods include findAndReplaceWorkspacePath, replaceDataSource, save, saveACopy, getExtent. |
| ListTableViews      | A Python List of TableView objects is returned. TableView object properties include datasetName, dataSource, definitionQuery, name, isBroken, and workspacePath. Methods include findAndReplaceWorkspacePath, and replaceDataSource.  |
| ListLayoutElements* | A Python List of layout element objects is returned. Layout element objects in the Python List can be DataFrame, GraphicElement, LegendElement, MapsurroundElement, PictureElement or TextElement objects.                            |
| ListBookmarks       | Returns a Python List of named tuples of bookmarks for a specified dataframe. The tuple contains the bookmark extent (extent object) and the bookmark name.                                                                           |

\*Code samples for listing, dataframes, layers, and layout elements are located in the `Map_production\Samples` folder.



## Managing layers

There are a number of functions in the `arcpy.mapping` module for manipulating and working with layers in a map document.

**arcpy.mapping functions for working with layers in the map document**

Table 9.4

| Function         | Description                                                                                                                                                                                        |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| AddLayer         | Adds a Layer object to the specified dataframe. The default placement option is similar to how the Add Data button works in ArcMap.                                                                |
| AddLayerToGroup  | Adds a Layer object to an existing group layer in the specified dataframe. The <code>isGroupLayer</code> property of a layer object can be used to confirm that the target layer is a group layer. |
| InsertLayer      | Inserts a Layer object to the specified dataframe, and is positioned relative to a specified existing layer.                                                                                       |
| ListBrokenLayers | Returns a Python list of Layer objects with broken data sources. Use this list function along with the Layer object method <code>replaceDataSource</code> to fix broken layers in the mxd.         |
| MoveLayer        | Moves a Layer in the dataframe relative to a specified existing layer. Similar to <code>InsertLayer</code> function.                                                                               |
| RemoveLayer      | Removes the specified layer from the specified dataframe.                                                                                                                                          |
| UpdateLayer      | Updates the properties for a layer based on a source layer or a .lyr file. Typically used to apply symbology from one layer to the specified layer.                                                |

Code samples for managing layers are located in the course's `Map_production\Samples` folder.



ArcGIS 10.1 Help  
*Updating and fixing data  
sources with  
arcpy.mapping*

There are two helper-type functions that can be used when updating contents of a map document that is loaded in memory:

1. The **arcpy.RefreshTOC()** function is used to view changes to the table of contents.
2. The **arcpy.RefreshActiveView()** function refreshes the active view and the table of contents in ArcMap.



Locate the desired layout element by its name property or its text value.

## Managing layout elements

When working with Layout elements in the map document, they must first be accessed with the `ListLayoutElements()` function. Once the desired element(s) are returned from the `ListLayoutElements()` function, the element properties can be modified and the map document can be saved.

An easy way to uniquely identify a layout element is to tag it with a name. This can be done in ArcMap in the layout view by selecting the layout element and then accessing the Size and Position properties and entering a unique Element Name. The Element Name becomes the element name tag.

For example, obtain a Python List of layout elements of a desired type (e.g., "TEXT\_ELEMENT"). Loop through the list looking for a particular text value or name tag. Once the tag or value is found, modify the element as needed.

Code samples for managing text and picture layout elements are located in the `Map_production\Samples` folder.

## Publishing maps

There are several functions available in the `arcpy.mapping` module and in the Server Toolbox > Publishing toolset for publishing a map service to ArcGIS Server.

A basic workflow for taking a map document to a published map service:

- Use `arcpy.mapping.MapDocument()` to obtain a reference to a map document.
- Use `arcpy.mapping.CreateMapSDDraft()` to create the service definition draft file (.sddraft).
- Use `arcpy.mapping.AnalyzeForSD()` to return any issue that need to be addressed.
- Stage the service definition draft using `arcpy.StageService_Server()` to create the service definition (.sd).
- Upload the service definition to ArcGIS Server with `arcpy.UploadServiceDefinition_Server()`.
- If you want to publish to ArcGIS Online, you can use `arcpy.SignInToPortal_Server()`.
- If you want to publish to ArcGIS Server, you can use an existing publisher connection file that can be created from the Catalog window in ArcMap.

A Server Definition Draft File (.sddraft) is a combination of the map document, information about the server connection, type of service being published, metadata for the service, a reference to the source data, and information about the map service (such as support for caching, feature access, OGC capabilities).

A Server Definition file contains all of the necessary information needed to publish the map service. This may also include a copy of the data if the option to copy data to the server is chosen.

The `arcpy.mapping.CreateMapSDDraft()` function creates the Server Definition Draft file and returns a Python dictionary of any errors and issues detected that need to be resolved prior to creating the Server Definition file.

The `arcpy.mapping.AnalyzeForSD()` function analyzes the Server Definition Draft file for any errors and performance issues that need to be addressed before creating the Service Definition file.



ArcGIS 10.1 Help Library:  
*CreateMapSDDraft*

The `arcpy.StageService_Server()` geoprocessing tool creates the Service Definition file prior to publishing.

The `arcpy.UploadServiceDefinition_Server()` geoprocessing tool uploads the service definition file to a specified server or to ArcGIS Online, based on the connection file.





ArcGIS 10.1 Help Library:  
*Exporting your map*



Specify the height and  
width using  
`df_export_width` and  
`df_export_height`.

Table 9.5

## Exporting maps and reports

There are several functions in the `arcpy.mapping` module used to export a data frame or page layout of a map document to supported graphics formats. There is also a function available for exporting your report to a file format.

### Exporting a map as a graphic

When exporting to graphics, consider that a dataframe does not have an associated page to provide height and width properties.

The following table shows the `arcpy.mapping` functions that can be used to export to supported graphics formats.

**arcpy.mapping functions**

| arcpy.mapping function | Graphics format                  |
|------------------------|----------------------------------|
| ExportToAI             | Adobe Illustrator                |
| ExportToBMP            | Windows Bitmap                   |
| ExportToEMF            | Enhanced Metafile                |
| ExportToEPS            | Encapsulated Postscript          |
| ExportToGIF            | Graphic Interchange              |
| ExportToJPEG           | Joint Photographic Experts Group |
| ExportToPDF*           | Portable Document Format         |
| ExportToPNG            | Portable Network Graphics        |
| ExportToSVG            | Scalable Vector Graphics         |
| ExportToTIFF           | Tagged Image File Format         |

\*Not all supported graphics formats support embedding of fonts. To ensure proper display of fonts and color models, such as RGB and CMYK, export to the PDF format.



ArcGIS 10.1 Help Library:

*ExportReport*

*(arcpy.mapping).*

## Exporting a report

A report that has been created and saved in ArcMap can be exported to a number of formats using the `arcpy.mapping.ExportReport()` function.

The `arcpy.mapping.ExportReport()` function exports a formatted tabular report created in ArcMap to a .txt, .xls, .html, .pdf, or .tif output file. The `ExportReport` function uses the data from the layer or table the report was authored from, along with the report layout file (.rlf) information from the ArcMap reporting tools, to output the report.

Numerous options are included in the `ExportReport` function to provide a report title, a starting page number, a page range to be exported, and an option for overriding the report definition query extent.

55 minutes



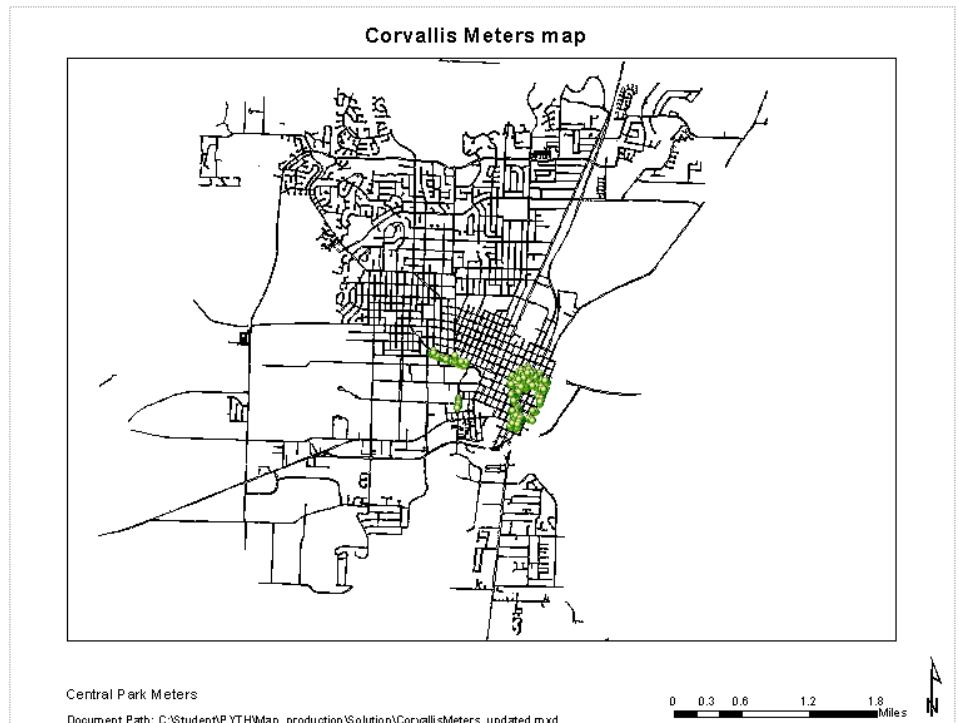
## Exercise 9: Work with map document contents

You have been tasked with updating the layers in a map document and will create a script to accomplish the task. The data source for the layers has been changed, so the layers must be updated in the map documents. The changes made need to be captured in the layout change history text element.

In this exercise, you will:

- Update layer properties in the map document.
- Update layout elements in the map document.
- Save changes to the map document.
- Verify the map document changes in ArcMap.

**Figure 9.1**  
Updated map document.



## Step 1: Access map document in ArcMap

Before making changes to the map document, you will view it in ArcMap.

- a Open ArcMap.
- b Open CorvallisMeters.mxd located in the C:\Student\PYTH\Map\_production folder.

There are four layers in the table of contents; ParkingMeters, Streets, Parks, and Building.

- c Click Layout View.

Notice the title and text elements of the map document. You will be updating the map document's layers, symbology, layout text elements, and title.

## Step 2: Access map document in script

The first step to automating the process of updating your map documents is to write a Python script. The Python script can be run outside of ArcGIS and can be easily modified to work with a list of .mxd files.

- a In PythonWin, create a new script named **MyUpdateMxd.py** and save it in the C:\Student\PYTH\Map\_production folder.

- b** Using the following workflow, add pseudocode to your script:
- Import the ArcPy mapping module.
  - Obtain reference to the `C:\Student\PYTH\Map_production\CorvallisMeters.mxd` file.
  - Create a dataframe object for the first dataframe in the map document.
  - Create a target layer object for the `ParkingMeters` layer.
  - Create a source layer object for the `C:\Student\PYTH\Map_production\ParkingMeters.lyr` layer file.
  - Update the target layer symbology from the source layer.
  - Create a layer object from the `C:\Student\PYTH\Map_production\Schools.lyr` layer file.
  - Add the layer to the map (take defaults).
  - Reorder the layers so that the `ParkingMeters` layer is ABOVE the `Schools` layer.
  - Update the map document title to 'Corvallis Meters Map'.
  - Obtain a list of layout text elements.
  - Iterate through each text element in the list.
  - If the element name is 'Corvallis Meters' update the element text value to 'Corvallis Parking Meters Inventory Report'.
  - Save a copy of the changes to a new map document.
  - Delete the map document object.
- c** In your script, add code to import the `arcpy.mapping` module as **MAP**.
- d** Use the `arcpy.mapping.MapDocument()` function to obtain a reference to `CorvallisMeters.mxd` and assign it to a variable named `mxd`.
- e** Use the `arcpy.mapping.ListDataFrames()` function on the `mxd` object to fetch the first dataframe.
- Hint:* You can append `[0]` to the end of the list function to return the first item in the list returned from the function:
- ```
arcpy.mapping.ListDataFrames(mxd)[0]
```
- f** Save your script and check for syntax errors.

### Step 3: Update layer symbology

Now that a reference to an existing map document has been obtained, the next step in the workflow is to update the symbology and placement of the `ParkingMeters` layer in the map document. You will also add the `Schools` layer file to the map document.

- a In your script, use the `arcpy.mapping.ListLayers()` function to create a layer object from the `ParkingMeters` map layer, and then assign it to the variable named `updateLayer`.  
*Hint: Refer to the workbook section on Managing layers for a code example.*
- b Use the `arcpy.mapping.Layer()` function to create a second layer object on the `ParkingMeters.lyr` file in the `Map_production` folder. Assign the object to the variable named `sourceLayer`.
- c Use the `arcpy.mapping.UpdateLayer()` function on the `updateLayer` to apply only the symbology from the `sourceLayer`.
- d Use the `arcpy.mapping.Layer()` function on the `Schools.lyr` file in the `Map_production` folder. Assign the object to the variable named `addLayer`.
- e Use the `arcpy.mapping.AddLayer()` function to add `addLayer` to the map document. Take all of the defaults.
- f From your research of the `AddLayer()` function, answer the following questions:

1. What are the position options for adding a layer to the map document?

---

2. If you need more precise control over the placement of the new layer than the `AddLayer` function provides, which `arcpy.mapping` function should you use?

---

Your final task in this step is to move the `ParkingMeters` layer to the top of the map document.

- g Use the `arcpy.mapping.ListLayers()` function to locate the newly added `Schools` layer in the dataframe and return the layer object to the variable named `refLayer`.
- h Use the `arcpy.mapping.MoveLayer()` function to move the `updateLayer` object to **BEFORE** the `refLayer` object.



Be sure to check the syntax for the `MoveLayer()` function in the Help documentation for specifying the correct order of the layers.

- i Save your script and check for syntax errors.

## Step 4: Update layout text elements

In your script, you have imported the required modules, obtained a reference to the map document, updated symbology for the ParkingMeters layer, added the Schools layer, and moved the ParkingMeters layer above the Schools layer.

In this step, you will add code to your script to update the map document title and update the layout 'Corvallis Meters' text element value.

- a In your script, update the map document's title to **'Corvallis Meters Map'**.  
*Hint: In the ArcGIS 10.1 Help Library, search for **map document properties (arcpy)** for appropriate syntax.*
- b Add code to return a Python List of layout text elements using the appropriate `arcpy.mapping` List function.
- c Loop through the list of layout text elements and locate the text element that is tagged with the name **'Corvallis Meters'**. Update the element text value to **'Corvallis Parking Meters Inventory Report'**.
- d Save a copy of the updated map document as **CorvallisMeters\_<your initials>.mxd** (e.g., **CorvallisMeters\_jb.mxd**).
- e Be sure to **del** the map document object in your script.
- f Save your script and check for syntax errors.
- g Run your script. Fix any errors that may occur at run time.
- h Close PythonWin upon successfully running of the script.

## Step 5: Verify changes in ArcMap

You have written a script to manipulate the contents of a map document and successfully run it outside of ArcMap. In this step, you will visually verify the changes in ArcMap.

- a In ArcMap, open **CorvallisMeters\_<your initials>.mxd**.

- b** In the data view, verify that the ParkingMeters layer is above the Schools layer and that the symbology for the ParkingMeters is a green orb.
- c** Switch to the layout view.
- d** Confirm that the map title text has been updated and the text in the lower-left corner of the layout has been updated.
- e** If you have time, you may proceed to the challenge step. (The challenge step should take approximately ten additional minutes.)
- f** If you would not like to complete the optional step, you may close ArcMap without saving changes.



## Challenge: Automate script for multiple mxds



10 minutes

- a Taking the script you just created, make the script dynamic by allowing the user to select a workspace that contains multiple map documents.
- b Update the script to support looping through a list of map documents obtained from this workspace and then process each map document based on the code in the original script.
- c After running the script successfully, create a custom script tool and test in ArcMap or ArcCatalog.



You can test your script tool using the C:\Student\PYTH\Map\_production workspace.

## Lesson review

1. When modifying a map document loaded in ArcMap, what method must be called to see the changes?

---

---

2. What changes can you make to a script to automate the map document update workflow?

---

---

---

3. What property must be changed in order to update the layout title text element?

---

---

## Answers to Lesson 9 questions

### Exercise 9: Work with map document contents (page 9-15)

1. What are the position options for adding a layer to the map document?

**AUTO\_ARRANGE, BOTTOM, TOP**

2. If you need more precise control over the placement of the new layer than the `AddLayer` function provides, which `arcpy.mapping` function should you use?

**`arcpy.mapping.InsertLayer()`**

## Challenge solution: Automate script for multiple mxds

Workflow for challenge:

- Use `arcpy.GetParameterAsText()` on hard-coded values in the script.
- Add one additional `arcpy.GetParameterAsText()` to pass a directory path to the script.
- Use the `arcpy.ListFiles()` function on the directory path to build a Python List of mxd file names.
- Add a `for-in` loop to iterate through the mxd file names just after the import code line.
- Modify the `arcpy.mapping.MapDocument()` function to accept the mxd file name passed by the `for-in` loop.

If you have time, create a script tool from your script and test in ArcMap or ArcCatalog.



# Esri data license agreement

## **IMPORTANT — READ CAREFULLY BEFORE OPENING THE SEALED MEDIA PACKAGE**

ENVIRONMENTAL SYSTEMS RESEARCH INSTITUTE, INC. (ESRI), IS WILLING TO LICENSE THE ENCLOSED ELECTRONIC VERSION OF THIS TRAINING COURSE TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS AND CONDITIONS CONTAINED IN THIS ESRI DATA LICENSE AGREEMENT. PLEASE READ THE TERMS AND CONDITIONS CAREFULLY BEFORE OPENING THE SEALED MEDIA PACKAGE. BY OPENING THE SEALED MEDIA PACKAGE, YOU ARE INDICATING YOUR ACCEPTANCE OF THE ESRI DATA LICENSE AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS AND CONDITIONS AS STATED, THEN ESRI IS UNWILLING TO LICENSE THE TRAINING COURSE TO YOU. IN SUCH EVENT, YOU SHOULD RETURN THE MEDIA PACKAGE WITH THE SEAL UNBROKEN AND ALL OTHER COMPONENTS (E.G., THE CD-ROM, TRAINING COURSE MATERIALS, TRAINING DATABASE, AS APPLICABLE) TO ESRI OR ITS AUTHORIZED INSTRUCTOR FOR A REFUND. NO REFUND WILL BE GIVEN IF THE MEDIA PACKAGE SEAL IS BROKEN OR THERE ARE ANY MISSING COMPONENTS.

## **ESRI DATA LICENSE AGREEMENT**

This is a license agreement, and not an agreement for sale, between you (Licensee) and Esri. This Esri data license agreement (Agreement) gives Licensee certain limited rights to use the electronic version of the training course materials, training database, software, and related materials (hereinafter collectively referred to as the "Training Course"). All rights not specifically granted in this Agreement are reserved to Esri and its licensor(s).

**Reservation of Ownership and Grant of License:** Esri and its licensor(s) retain exclusive rights, title, and ownership to the copy of the Training Course licensed under this Agreement and hereby grant to Licensee a personal, nonexclusive, nontransferable license to use the Training Course as a single package for Licensee's own personal use only pursuant to the terms and conditions of this Agreement. Licensee agrees to use reasonable efforts to protect the Training Course from unauthorized use, reproduction, distribution, or publication.

**Proprietary Rights and Copyright:** Licensee acknowledges that the Training Course is proprietary and confidential property of Esri and its licensor(s) and is protected by United States copyright laws and applicable international copyright treaties and/or conventions.

### Permitted Uses:

- Licensee may run the setup and install one (1) copy of the Training Course onto a permanent electronic storage device and reproduce one (1) copy of the Training Course and/or any online documentation in hard-copy format for Licensee's own personal use only.
- Licensee may use one (1) copy of the Training Course on a single processing unit.
- Licensee may make only one (1) copy of the original Training Course for archival purposes during the term of this Agreement, unless the right to make additional copies is granted to Licensee in writing by Esri.
- Licensee may use the Training Course provided by Esri for the stated purpose of Licensee's own personal GIS training and education.

### Uses Not Permitted:

- Licensee shall not sell, rent, lease, sublicense, lend, assign, time-share, or transfer, in whole or in part, or provide unlicensed third parties access to the Training Course, any updates, or Licensee's rights under this Agreement.
- Licensee shall not separate the component parts of the Training Course for use on more than one (1) computer, used in conjunction with any other software package, and/or merged and compiled into a separate database(s) for other analytical uses.
- Licensee shall not reverse engineer, decompile, or disassemble the Training Course, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this restriction.
- Licensee shall not make any attempt to circumvent the technological measure(s) (e.g., software or hardware key) that effectively controls access to the Training Course, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this restriction.
- Licensee shall not remove or obscure any copyright, trademark, and/or proprietary rights notices of Esri or its licensor(s).

**Term:** The license granted by this Agreement shall commence upon Licensee's receipt of the Training Course and shall continue until such time that (1) Licensee elects to discontinue use of the Training Course and terminates this Agreement or (2) Esri terminates for Licensee's material breach of this Agreement. The Agreement shall automatically terminate without notice if Licensee fails to comply with any provision of this Agreement. Upon termination of this Agreement in either instance, Licensee shall return to Esri or destroy all copies of the Training Course, and any whole or partial copies, in any form and deliver evidence of such destruction to Esri, which evidence shall be in a form acceptable to Esri in its sole discretion. The parties hereby agree that all provisions that operate to protect the rights of Esri and its licensor(s) shall remain in force should breach occur.

**Limited Warranty and Disclaimer:** Esri warrants that the media upon which the Training Course is provided will be free from defects in materials and workmanship under normal use and service for a period of ninety (90) days from the date of receipt.

EXCEPT FOR THE LIMITED WARRANTY SET FORTH ABOVE, THE TRAINING COURSE CONTAINED THEREIN IS PROVIDED "AS-IS," WITHOUT WARRANTY OF ANY KIND, EITHER

EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT. ESRI DOES NOT WARRANT THAT THE TRAINING COURSE WILL MEET LICENSEE'S NEEDS OR EXPECTATIONS; THAT THE USE OF THE TRAINING COURSE WILL BE UNINTERRUPTED; OR THAT ALL NONCONFORMITIES, DEFECTS, OR ERRORS CAN OR WILL BE CORRECTED. THE TRAINING DATABASE HAS BEEN OBTAINED FROM SOURCES BELIEVED TO BE RELIABLE, BUT ITS ACCURACY AND COMPLETENESS, AND THE OPINIONS BASED THEREON, ARE NOT GUARANTEED. THE TRAINING DATABASE MAY CONTAIN SOME NONCONFORMITIES, DEFECTS, ERRORS, AND/OR OMISSIONS. ESRI AND ITS LICENSOR(S) DO NOT WARRANT THAT THE TRAINING DATABASE WILL MEET LICENSEE'S NEEDS OR EXPECTATIONS, THAT THE USE OF THE TRAINING DATABASE WILL BE UNINTERRUPTED, OR THAT ALL NONCONFORMITIES CAN OR WILL BE CORRECTED. ESRI AND ITS LICENSOR(S) ARE NOT INVITING RELIANCE ON THIS TRAINING DATABASE, AND LICENSEE SHOULD ALWAYS VERIFY ACTUAL DATA, WHETHER MAP, SPATIAL, RASTER, TABULAR INFORMATION, AND SO FORTH. THE DATA CONTAINED IN THIS PACKAGE IS SUBJECT TO CHANGE WITHOUT NOTICE.

**Exclusive Remedy and Limitation of Liability:** During the warranty period, Licensee's exclusive remedy and Esri's entire liability shall be the return of the license fee paid for the Training Course upon the Licensee's deinstallation of all copies of the Training Course and providing a Certification of Destruction in a form acceptable to Esri.

IN NO EVENT SHALL ESRI OR ITS LICENSOR(S) BE LIABLE TO LICENSEE FOR COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, LOST PROFITS, LOST SALES OR BUSINESS EXPENDITURES, INVESTMENTS, OR COMMITMENTS IN CONNECTION WITH ANY BUSINESS, LOSS OF ANY GOODWILL, OR FOR ANY INDIRECT, SPECIAL, INCIDENTAL, AND/OR CONSEQUENTIAL DAMAGES ARISING OUT OF THIS AGREEMENT OR USE OF THE TRAINING COURSE, HOWEVER CAUSED, ON ANY THEORY OF LIABILITY, AND WHETHER OR NOT ESRI OR ITS LICENSOR(S) HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. THESE LIMITATIONS SHALL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY.

**No Implied Waivers:** No failure or delay by Esri or its licensor(s) in enforcing any right or remedy under this Agreement shall be construed as a waiver of any future or other exercise of such right or remedy by Esri or its licensor(s).

**Order for Precedence:** This Agreement shall take precedence over the terms and conditions of any purchase order or other document, except as required by law or regulation.

**Export Regulation:** Licensee acknowledges that the Training Course and all underlying information or technology may not be exported or re-exported into any country to which the U.S. has embargoed goods, or to anyone on the U.S. Treasury Department's list of Specially Designated Nationals, or to the U.S. Commerce Department's Table of Deny Orders. Licensee shall not export the Training Course or any underlying information or technology to any facility in violation of these or other applicable laws and regulations. Licensee represents and warrants that it is not a national or resident of, or located in or under the control of, any country subject to such U.S. export controls.

**Severability:** If any provision(s) of this Agreement shall be held to be invalid, illegal, or unenforceable by a court or other tribunal of competent jurisdiction, the validity, legality, and enforceability of the remaining provisions shall not in any way be affected or impaired thereby.

**Governing Law:** This Agreement, entered into in the County of San Bernardino, shall be construed and enforced in accordance with and be governed by the laws of the United States of America and the State of California without reference to conflict of laws principles.

**Entire Agreement:** The parties agree that this Agreement constitutes the sole and entire agreement of the parties as to the matter set forth herein and supersedes any previous agreements, understandings, and arrangements between the parties relating hereto.



# B

## Answers to lesson review questions

### Lesson 1: Running scripts in Python (page 1-21)

1. How can you access geoprocessing functionality in a Python script?

**Import the ArcPy site package.**

2. What is ArcPy?

**ArcPy provides Python access to all geoprocessing tools and a wide variety of useful functions and classes for working with and interrogating GIS data.**

3. In the Python window, how can you view help for a geoprocessing tool?

**Type help, followed by an open parenthesis, followed by the tool name, followed by a close parenthesis. The tool syntax and help documentation will display in the Python window.**

**Example: `help(arcpy.Buffer_analysis)`**

## Lesson 2: Describing data (page 2-22)

1. When describing a feature class, what properties can be useful in a script?

**Answers might include:**

- **featureType**
- **hasM**
- **hasZ**
- **shapeFieldName**
- **shapeType**
- **spatialReference**
- **extent**
- **fields**

2. What is returned by the extent property of a dataset?

**An extent object. The extent object contains properties for Min and Max X, Min and Max Y, height, width, and M and Z values.**

3. How can you obtain the name of a spatialReference for a feature class?

**Describe the feature class. In the Describe object properties, access the name property of the spatialReference object property.**

## Lesson 3: Automating scripts with lists (page 3-16)

1. What do all ArcPy List functions return?

**A list of items that contains either string values or objects.**

2. How do you iterate through a list returned by the List function?

- **A for-in loop**
- **A while loop**

3. Write a line of code that returns a list of integer fields from a table.

```
fieldList = [field.name for field in arcpy.ListFields("Addresses")  
             if field.type == "Integer"]
```

## Lesson 4: Working with Selections (page 4-20)

1. List two geoprocessing tools that must use a feature layer to create selections.

**The `arcpy.SelectLayerByLocation` tool and the `arcpy.SelectLayerByAttribute` tool must use a feature layer to create selections.**

2. What kind of schema changes can you make on a field with a `FieldInfo` object?

**You can make the following schema changes on a field with a `FieldInfo` object:**

- **Rename the field**
- **Change the field visibility to `HIDDEN` or `VISIBLE`**
- **Set a split rule to `NONE` or `RATIO`**

3. If a field is set to `HIDDEN` in a `FieldInfo` object, will the field be available in the output feature layer?

**No. When the `FieldInfo` object is used with the `arcpy.MakeFeatureLayer` tool, the field will not be present in the output feature layer.**

## Lesson 5: Working with Cursors (page 5-20)

1. When working with the da cursors, which Python statement can be used to automatically close the cursor?

**c. A with loop**

2. Which token can be used with the da cursor to efficiently return the X&Y values of a point feature?

**SHAPE@XY token, SHAPE@X along with SHAPE@Y, or SHAPE@TRUECENTROID**

3. Write a single line of code to return a sorted cursor of descending LAND\_VALUE values from the Corvallis Parcel feature class.

```
cur = [row for row in sorted(arcpy.da.SearchCursor(
    "Parcel", "LAND_VALUE"), reverse = True)]
```

## Lesson 6: Working with Geometry objects (page 6-29)

1. To create a Polyline or Polygon geometry object, what two objects must first be created?

**A Point object and an Array object must be created before creating a Polyline or Polygon geometry object.**

2. List some geometry tokens that efficiently access Shape geometry properties.

**SHAPE@XY, SHAPE@AREA, SHAPE@LENGTH, SHAPE@TRUECENTROID**

3. Provide a couple of examples in which you would use a geometry object with a geoprocessing tool.

**Possible examples may include:**

- **Using a Polygon geometry object with the Clip tool.**
- **Using a Polygon geometry object with the Overlay tools.**
- **Creating a new buffer feature from a Polyline or Polygon geometry object or from a Point object.**

## Lesson 7: Sharing scripts (page 7-29)

1. How can you make a Python script dynamic?

**Use the `arcpy.GetParameterAsText()` function.**

2. What are the steps needed to share a script as a geoprocessing package?

**Create script tool, run script tool as a dialog or in the Python window, edit the Item Description to add the Summary and Tags (required at a minimum), share result as a geoprocessing package.**

3. List the basic workflow for sharing GIS workflows with others.

**Author, debug and test script, create custom script tool, edit Item Description, run script tool in ArcGIS Desktop, verify results support scenario, share results as geoprocessing package.**

## Lesson 8: Debugging scripts and handling runtime errors (page 8-23)

1. Of the error handling techniques discussed in the class, which technique returns the most amount of detail?

**The traceback module used in conjunction with the sys module**

2. Which error handlers should you use if *only* geoprocessing errors are to be handled?

**The arcpy.ExecuteError and arcpy.ExecuteWarning handlers will raise an exception only if a geoprocessing tool error or warning is encountered.**

3. Write an except block that adds a custom error message to the Progress dialog box, the Python window, the Results, and the Interactive Window in PythonWin.

```
except:
    msg = "My custom error message"
    arcpy.AddError(msg)
    print msg
```

4. What functionality is available in PythonWin for stepping through your code?

**The Debugging toolbar**



## Lesson 9: Automating map production (page 9-22)

1. When modifying a map document loaded in ArcMap, what method must be called to see the changes?

**`arcpy.RefreshActiveView()` or `arcpy.RefreshTOC()`**

2. What changes can you make to a script to automate the map document update workflow?

- Use `arcpy.ListFiles()` function to return a Python List of mxds and iterate through the list.
- Replace all hardcoded variables with `arcpy.GetParameterAsText()`.
- Add script to custom toolbox, then run the custom tool in Batch mode.

3. What property must be changed in order to update the layout title text element?

**The change must be made to the map document's title property—*not* to the layout text element.**